i-Tree

Hydro+

Technical Manual

09.02.2020

v6.3 beta

i-Tree is a cooperative initiative

About

Introduction

This document is intended to facilitate development and use of the Hydro+ environmental model suite – referred to as UnifiedHydro 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 Hydro+, 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 Hydro+ is part of the i-Tree Research Suite, i-Tree Tools' usual level of technical support and documentation is not available for Hydro+. 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

Getting Started

As a first step to using Hydro+ for your own projects, we recommend using Hydro+ Test Cases to begin familiarizing yourself with the available models and settings. Test Cases are maintained as part of Hydro+ 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:
`…\UnifiedHydro\TestingFilesAndScript\TestCases\`

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

- Hydro+ 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 Hydro+ users, please choose a Test Case of interest from the SOP for Testing Code then proceed through the Inputs & Running Hydro+ 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 parameters: review to Abdi 2019 in the Hydro science archives (section 5.2 details GI methods, Tables 13 & 17 describe parameters used in GI TestCases); SWMM LID documentation; and the Hydro v6 User Manual.
4. Save config file & run UnifiedHydro.exe same as without GI (Inputs & Running Hydro+ Models).
5. Access outputs as you would without GI (Outputs).

Inputs & Running Hydro+ Models

Introduction to Using Hydro+

To run any of the models within Hydro+, the UnifiedHydro 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:\0UnifiedHydro\x64\Release\0UnifiedHydro.exe C:\0UnifiedHydro\Inputs
```

The program will look in the inputs directory for a configuration file “UnifiedHydroConfig.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. Hydro+ 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:

- UnifiedHydroConfig.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 Hydro+
  - 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:

- UnifiedHydroConfig.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 (UnifiedHydroConfig.xml)

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 UnifiedHydroConfig.xml file embedded below is from one of the Test Cases 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.

UnifiedHydroConfig.xml

For more information about Hydro+ 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 …\UnifiedHydro\TestingFilesAndScript\TestCases\StatHydro\powIR_defaultParams\input\UnifiedHydroConfig.xml

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 UnifiedHydroConfig.xml GI features can be enabled by adding DataFolders with Type set to a GI structure type. Each DataFolder can describe the aggregate & average statistics representing all GI structures of a certain type (a 'lumped' or 'semi-distributed' representation, as is done by the Hydro GUI and for the 'bulk area' of the StatisticalHydro model), or each DataFolder can represent an individual GI structure. For more guidance on GI parameterization: explore commented example GI config file (Hydro+\UnifiedHydro\TestingFilesAndScript\TestCases\StatHydro\GI\UnifiedHydroConfig_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.

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.

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. **This 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.

Acquisition of raw weather data

For each year including data of interest:

1) Select a weather station (make note of its WBAN identification #) and year of data for your use.
2) Go to ftp://ftp.ncei.noaa.gov/pub/data/noaa/ in your web browser.
3) Enter the folder for the year of data you are interested in.
4) Select the .gz file for the weather station you are interested in. File names are formatted with weather station identifiers and data year: <USAF#>-<WBAN#>-<YYYY>.gz
5) Extract raw, unformatted weather data from the .gz archive file, using 7-Zip or similar program.
6) Format raw, unformatted weather data that you extracted using ishapp2.exe (provided by NOAA) and command format: ishapp2.exe [input file] [output file]

Preprocessing weather data

Now that you have raw, formatted hourly weather data for a year of interest, run it in the i-Tree Research Suite's weather preprocessor utility:

1) Start the “WeatherDataExtractor.exe” program. This is an unpublished EXE which your operating system will not recognize, so we expect you may get a cautionary warning about opening it. We affirm this file is safe, and the file integrity can be checked using checksums.
2) Enter the LocationSpecies.mdb file ***

Multi-year weather data

For multiple years of weather data, perform the above steps for each year individually, and then append the time periods of interest together in an un-interrupted timeseries that matches your intended simulation period.

Outputs

After you’ve run the simulations you are interested in, find outputs in the OutputDirectory (defined in UnifiedHydroConfig.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 UnifiedHydroConfig.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 Hydro+ 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\UnifiedHydro
    - Not OK: C:\Users\coviller\Dev Files\SVN\hydrologyv6\branches\UnifiedHydro

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 Hydro+ R&D and need access to SVN, please register a new account at https://code.itreetools.org and notify project leaders Robert.Coville@davey.com and 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

UnifiedHydro – the 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

Buffer – the model for identifying nutrient hotspots – is version controlled on git at https://code.itreetools.org/Users/rcoville/Buffer

Chill – the model for mechanistically predicting heating & cooling energy saving benefits – is version controlled on git at https://code.itreetools.org/Users/rcoville/Chill

Concepts and Definitions

The TortoiseSVN manual is useful reference: https://tortoisesvn.net/docs/release/TortoiseSVN_en/index.html Some definitions below are copied from that reference.

Trunk

This should be the location the main code base is stored, whether that's stable or in development. In UnifiedHydro's repository (hydrologyv6) the trunk is defunct (tree architecture problems prevent its use) and work has continued only on branches and tags as recent as October 2018.

Branches

A term frequently used in revision control systems to describe what happens when development forks at a particular point and follows 2 separate paths. You can create a branch off the main development line so as to develop a new feature without rendering the main line unstable. Or you can branch a stable release to which you make only bug fixes, while new developments take place on the unstable trunk.

Tags

Tags are checkpoints in development, usually to isolate a stable version of the code for production. For example: a revision in the hydrologyv6 repository is tagged "HydrologyEXE_For_HydroV6.1.0b" indicating that this is the version of the code used to produce the Hydrology.exe file running in Hydro v6.1.0 beta.

Working Copy

This is your local "sandbox", the area where you work on the versioned files, and it normally resides on your local hard disk. You create a working copy by doing a "Checkout" from a repository, and you feed your changes back into the repository using "Commit".

Updating

This Subversion command pulls down the latest changes from the repository into your working copy, merging any changes made by others with local changes in the working copy.

Show Log

Show the revision history of a file or folder. Also known as "History". The log can be used to isolate what revisions affected a certain file, or what files a specific revision changed. In the "Show Log" window, right-clicking on a specific file in a revision's change details gives the option to compare that revision's file version with the base (working copy) file version.

Adding

"Add" is a Subversion command that is used to add a file or directory to your working copy. The new items are added to the repository when you commit.

Merging

The process by which changes from the repository are added to your working copy without disrupting any changes you have already made locally. Sometimes these changes cannot be reconciled automatically and the working copy is said to be in conflict.

Merging happens automatically when you update your working copy. You can also merge specific changes from another branch using TortoiseSVN's Merge command.

Reverting

Subversion keeps a local “pristine” copy of each file as it was when you last updated your working copy. If you have made changes and decide you want to undo them, you can use the “revert” command to go back to the pristine copy.

This can also be used after errors have been committed; changes from specific past revisions can be reverted, undoing only the changes for those reverted revisions without losing progress from other revisions.

Conflicts

When changes from the repository are merged with local changes, sometimes those changes occur on the same lines. In this case Subversion cannot automatically decide which version to use and the file is said to be in conflict. You have to edit the file manually and resolve the conflict before you can commit any further changes.

Procedures

Checkout

<placeholder for more details – for now, see “Concepts”>

Update

<placeholder for more details – for now, see “Concepts”>

Commit

<placeholder for more details – for now, see “Concepts”>

Merge

<placeholder for more details – for now, see “Concepts”>

Find specific change/revision

<placeholder for more details – for now, see “Concepts”>

SOP for Testing Code

A test script and sample inputs and expected outputs are setup to streamline model testing. The standard test procedure will compile a working copy of the UnifiedHydro code and run it against the following scenarios (as of August 23, 2019 revision 427 of hydrologyv6 branch) defined in \UnifiedHydro\TestingFilesAndScript\ListOfTests.txt:

- **Statistical Hydro model**:
    - **1.** Power Infiltration-Rate Decay mode; Rock Creek Basin sample project inputs, with powdecayTI.dat as an input and CalibrationInterval=NoCalibration.
    - **2.** Power Infiltration-Rate Decay mode; Rock Creek Basin sample project inputs, with no TI file present and only a DEM.dat, and CalibrationInterval=Weekly. For stable Hydro+, this test case matches Hydro v6.3's Sample Project settings precisely.
    - **3.** Exponential Infiltration-Rate Decay mode; Rock Creek Basin sample project inputs, with preloaded expdecayTI.dat from Hydro v6.1.3 GUI, and CalibrationInterval=Hourly
    - **4.** GI: Green infrastructure types each have one test case, based on SWMM LID samples: bioRetention, infilTrench, porousPavement, rainGarden, swale. More in development.
- **Spatial Temperature model** (temporary as 2018 test cases are prepared for field comparison): Baltimore City inputs using 2015 weather data preprocessed by Ted before November 2018, with the following output settings:
    - **5.** Example B: print select block group hourly & daily outputs for the simulation period.
    - **6.** Example D: print select cell-based (Row-Col) hourly outputs for the simulation period.
    - **7.** Example E: print timestep-based raster outputs for a select time period.

About the Statistical Hydro model Test Cases from i-Tree Hydro v6.3:

The StatHydro Test Cases are based on the i-Tree Hydro GUI v6.3 Sample Project in the Rock Creek watershed. The Rock Creek watershed is located in Washington, DC and has a delineated watershed area of 161.44 sq. km. The USGS gauge, from which the discharge data was retrieved, is USGS 01648000. The weather data was retrieved from a station approximately 15 kilometers south of the USGS gauge, at Washington National Airport with a USAF-WBAN weather station ID of 724050-13743. Both the discharge data and weather data range from approximately 01/01/2010 12:00 am to 12/29/2010 11:00 pm. The fraction of impervious area and pervious area is 33.3% and 66.7% respectively; the percent tree cover is 41.3%. The average annual precipitation is 39.7 inches of rain and 15.4 inches of snow; the average temperature in winter (Dec-Jan-Feb) is a low of 30.7 F and a high of 45.8 F, while the average temperature in the summer (Jun-Jul-Aug) is a low of 69.0 F and a high of 86.4 F.

Procedure for Running Test Script

Step-by-step screenshots of this process are available in the attached file:

Recording_20181115_2335.mht

1. Open the UnifiedHydro solution (e.g. https://svn.itreetools.org/svn/hydrologyv6/branches/UnifiedHydro/UnifiedHydro/UnifiedHydro.sln) in Visual Studio 2019.
2. Set the solution configuration to "Release" mode and the solution platform to "Win32" (usually it will be in Debug, Win32 mode during development).

3. Go to Build > Rebuild Solution to compile a fresh copy of the UnifiedHydro.exe. The Output window in Visual Studio should show compilation processes and will end by stating if the solution compiled successfully or failed.
4. Find the /Release/ folder in the same directory as the UnifiedHydro.sln file, and in that /Release/ folder confirm there is a UnifiedHydro.exe created as recently as expected.
5. Go to the /TestingFilesAndScript/ directory (e.g. https://svn.itreetools.org/svn/hydrologyv6/branches/UnifiedHydro/TestingFilesAndScript) and open ListOfTests.txt.
6. Confirm that the line defining the Executable location points to the correct location for the compiled UnifiedHydro.exe to be tested. This should be the case by default (via ListOfTests.txt including "Executable,..\UnifiedHydro\Release\UnifiedHydro.exe") so that steps 5 & 6 of this SOP require no action.
7. Run the Python script "TestToCompareUHOutputToStandards.py" from a command prompt or your preferred Python IDE.
    a. Nothing in this file or standard inputs should need to be changed; as of rev292, the script will use the UnifiedHydro.exe that's in /TestingFilesAndScript/ along with the various input and expected output sets to test that exe against standard results, automagically ensuring config files are set to appropriate input/output paths.
    b. If each simulation ends with "Press enter to exit" (as it currently does so users can view the terminal output at run completion before closing it), then you'll need to press enter in the terminal between each test case. It may take a few "enters" before the next case begins. To fix this we can write in the output directory a Log.dat of the terminal output for user review if needed, then just close the terminal at simulation completion so the test script can run through all test cases with no further input.
8. 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.
9. If outputs are only slightly different that may be acceptable, or if certain expected outputs are not yet reliable (i.e. block group outputs in Example B of rev292) then 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.
10. 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 UnifiedHydro 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.

Git-based Project development

Utilities and models related to Hydro+ are developed and stored on git repositories. Below is an example workflow for development of code on one of our git repositories.

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.

Development Notes

Config file modifications

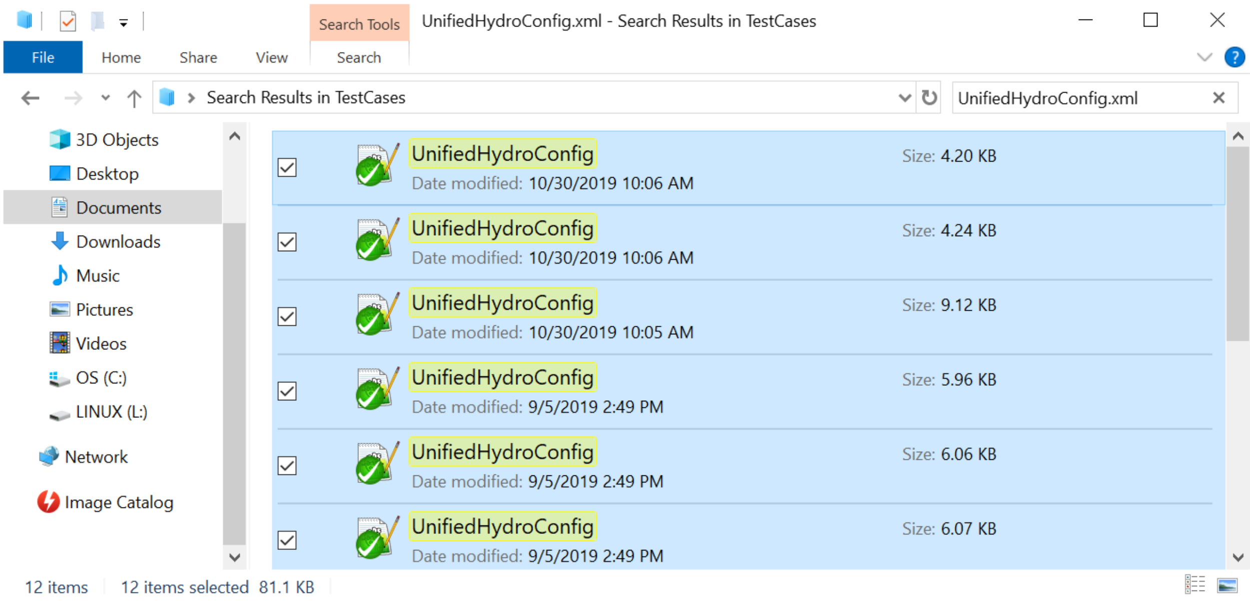
The config files included in /TestingFilesAndScript/TestCases/ serve as 'master copy' config files; as we update Hydro+, users can compare older config files with the latest in TestCases to get their files up-to-speed with the latest config file features.

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.

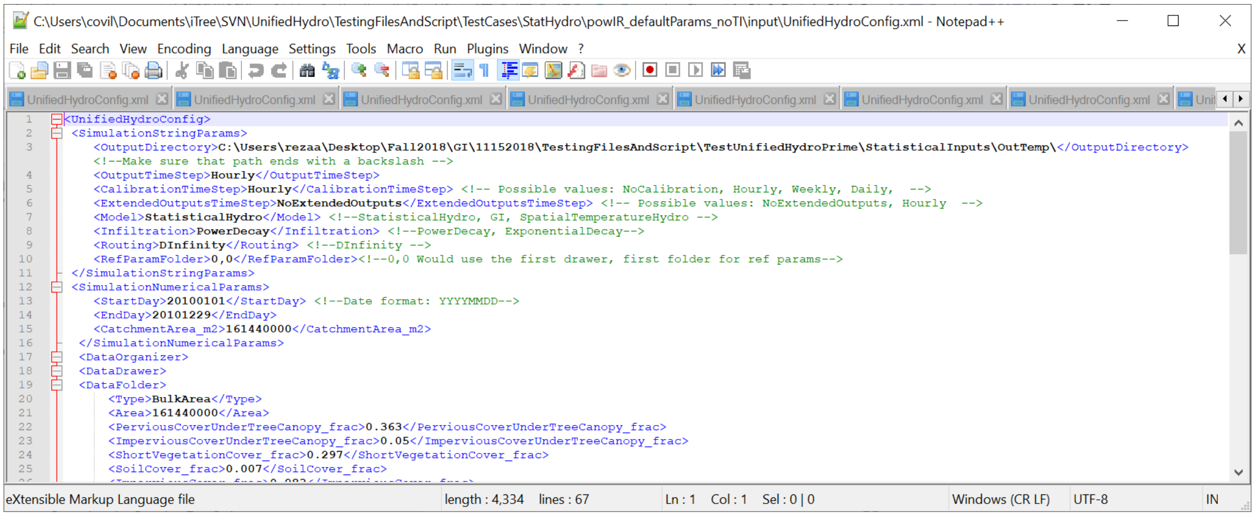
Batch superficial changes (file only, no code changes)

1. Open all UnifiedHydroConfig.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.



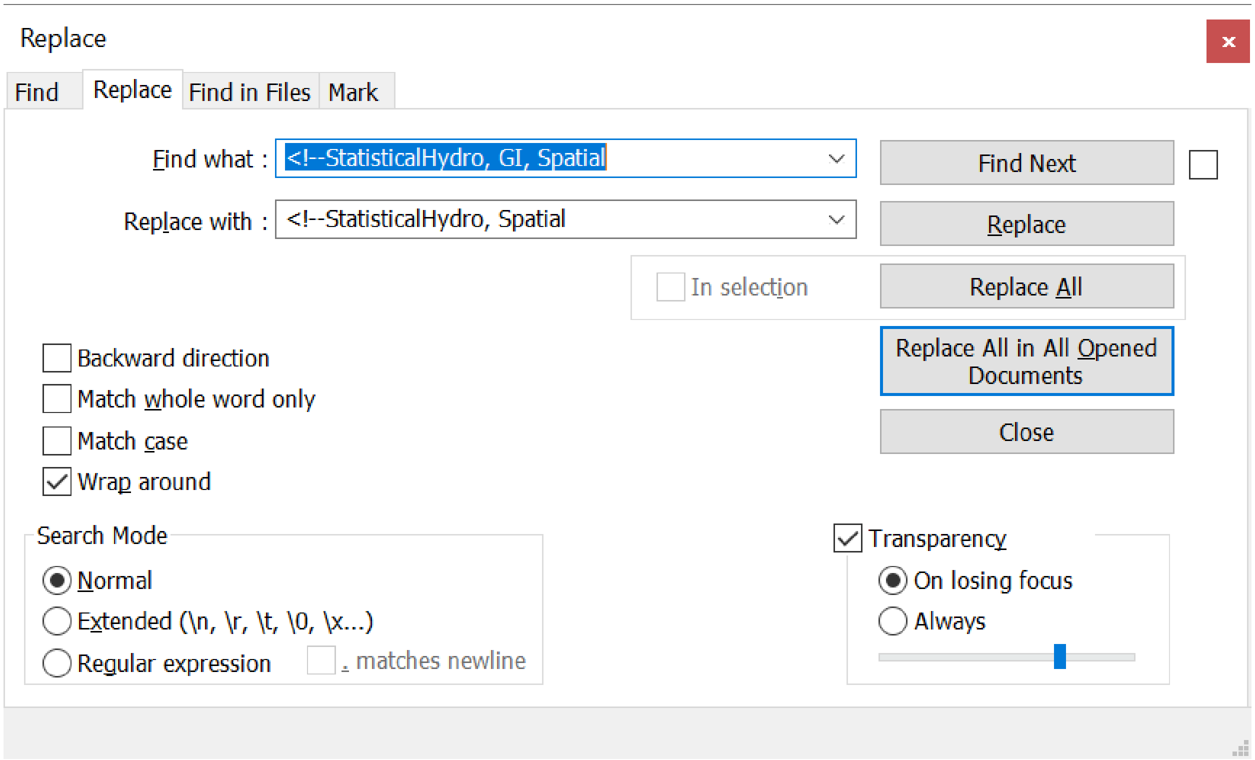
	Name	Date modified	Size
☑	UnifiedHydroConfig	10/30/2019 10:06 AM	4.20 KB
☑	UnifiedHydroConfig	10/30/2019 10:06 AM	4.24 KB
☑	UnifiedHydroConfig	10/30/2019 10:05 AM	9.12 KB
☑	UnifiedHydroConfig	9/5/2019 2:49 PM	5.96 KB
☑	UnifiedHydroConfig	9/5/2019 2:49 PM	6.06 KB
☑	UnifiedHydroConfig	9/5/2019 2:49 PM	6.07 KB


C:\Users\covil\Documents\iTree\SVN\UnifiedHydro\TestingFilesAndScript\TestCases\StatHydro\powIR_defaultParams_noTI\input\UnifiedHydroConfig.xml - Notepad++

```
<UnifiedHydroConfig>
 <SimulationStringParams>
    <OutputDirectory>C:\Users\rezaa\Desktop\Fall2018\GI\11152018\TestingFilesAndScript\TestUnifiedHydroPrime\StatisticalInputs\OutTemp\</OutputDirectory>
    <!--Make sure that path ends with a backslash -->
    <OutputTimeStep>Hourly</OutputTimeStep>
    <CalibrationTimeStep>Hourly</CalibrationTimeStep> <!-- Possible values: NoCalibration, Hourly, Weekly, Daily,  -->
    <ExtendedOutputsTimeStep>NoExtendedOutputs</ExtendedOutputsTimeStep> <!-- Possible values: NoExtendedOutputs, Hourly  -->
    <Model>StatisticalHydro</Model> <!--StatisticalHydro, GI, SpatialTemperatureHydro -->
    <Infiltration>PowerDecay</Infiltration> <!--PowerDecay, ExponentialDecay-->
    <Routing>DInfinity</Routing> <!--DInfinity -->
    <RefParamFolder>0,0</RefParamFolder><!--0,0 Would use the first drawer, first folder for ref params-->
 </SimulationStringParams>
 <SimulationNumericalParams>
    <StartDay>20100101</StartDay> <!--Date format: YYYYMMDD-->
    <EndDay>20101229</EndDay>
    <CatchmentArea_m2>161440000</CatchmentArea_m2>
  </SimulationNumericalParams>
  <DataOrganizer>
  <DataDrawer>
  <DataFolder>
      <Type>BulkArea</Type>
      <Area>161440000</Area>
      <PerviousCoverUnderTreeCanopy_frac>0.363</PerviousCoverUnderTreeCanopy_frac>
      <ImperviousCoverUnderTreeCanopy_frac>0.05</ImperviousCoverUnderTreeCanopy_frac>
      <ShortVegetationCover_frac>0.297</ShortVegetationCover_frac>
      <SoilCover_frac>0.007</SoilCover_frac>
```

eXtensible Markup Language file   length : 4,334   lines : 67   Ln : 1   Col : 1   Sel : 0 | 0   Windows (CR LF)   UTF-8   IN

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 this example, the comment about available Model tag values is being changed to remove GI (since the GI model mode is now built into StatisticalHydro model mode)



Find what : <!--StatisticalHydro, GI, Spatial

Replace with : <!--StatisticalHydro, Spatial

☐ In selection

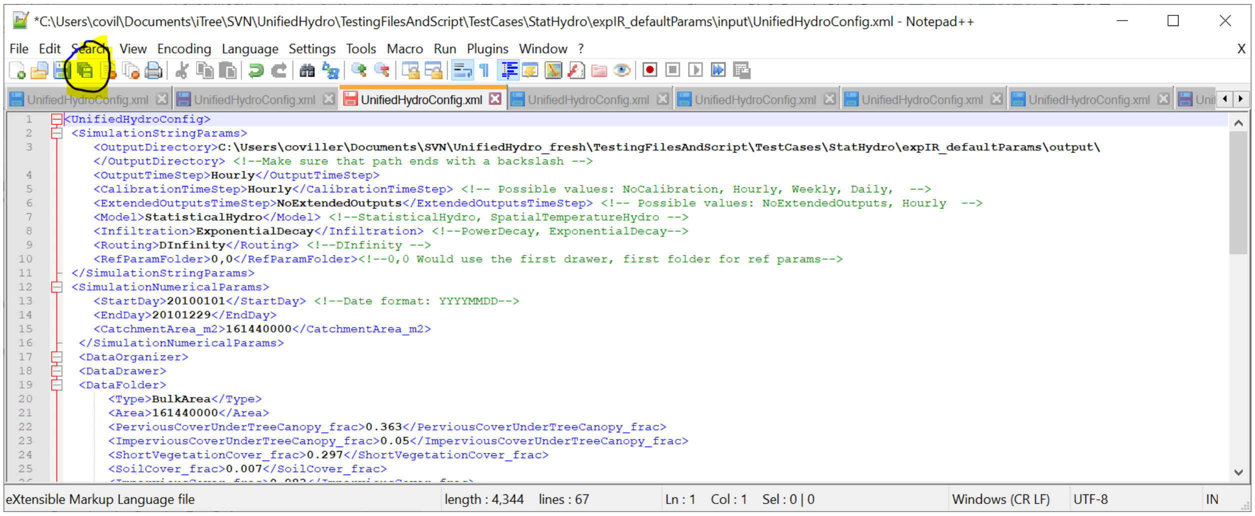
☐ Backward direction
☐ Match whole word only
☐ Match case
☑ Wrap around

Search Mode
● Normal
○ Extended (\n, \r, \t, \0, \x...)
○ Regular expression ☐ . matches newline

☑ Transparency
● On losing focus
○ Always

Be careful to use a detailed enough “Find” value 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


*C:\Users\covil\Documents\iTree\SVN\UnifiedHydro\TestingFilesAndScript\TestCases\StatHydro\expIR_defaultParams\input\UnifiedHydroConfig.xml - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
UnifiedHydroConfig.xml
eXtensible Markup Language file
length : 4,344 lines : 67
Ln : 1 Col : 1 Sel : 0 | 0
Windows (CR LF)
UTF-8
IN

```xml
<UnifiedHydroConfig>
 <SimulationStringParams>
    <OutputDirectory>C:\Users\coviller\Documents\SVN\UnifiedHydro_fresh\TestingFilesAndScript\TestCases\StatHydro\expIR_defaultParams\output\
    </OutputDirectory> <!--Make sure that path ends with a backslash -->
    <OutputTimeStep>Hourly</OutputTimeStep>
    <CalibrationTimeStep>Hourly</CalibrationTimeStep> <!-- Possible values: NoCalibration, Hourly, Weekly, Daily,  -->
    <ExtendedOutputsTimeStep>NoExtendedOutputs</ExtendedOutputsTimeStep> <!-- Possible values: NoExtendedOutputs, Hourly  -->
    <Model>StatisticalHydro</Model> <!--StatisticalHydro, SpatialTemperatureHydro -->
    <Infiltration>ExponentialDecay</Infiltration> <!--PowerDecay, ExponentialDecay-->
    <Routing>DInfinity</Routing> <!--DInfinity -->
    <RefParamFolder>0,0</RefParamFolder><!--0,0 Would use the first drawer, first folder for ref params-->
 </SimulationStringParams>
 <SimulationNumericalParams>
    <StartDay>20100101</StartDay> <!--Date format: YYYYMMDD-->
    <EndDay>20101229</EndDay>
    <CatchmentArea_m2>161440000</CatchmentArea_m2>
  </SimulationNumericalParams>
  <DataOrganizer>
  <DataDrawer>
  <DataFolder>
      <Type>BulkArea</Type>
      <Area>161440000</Area>
      <PerviousCoverUnderTreeCanopy_frac>0.363</PerviousCoverUnderTreeCanopy_frac>
      <ImperviousCoverUnderTreeCanopy_frac>0.05</ImperviousCoverUnderTreeCanopy_frac>
      <ShortVegetationCover_frac>0.297</ShortVegetationCover_frac>
      <SoilCover_frac>0.007</SoilCover_frac>
```

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 UnifiedHydro.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 UH 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)

UnifiedHydro 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 UnifiedHydro.exe (freshly compiled in Release mode from https://code.itreetools.org/svn/hydrologyv6/branches/UnifiedHydro) in …\LCscaling\AutoRun\HydroBin\
3. Compile AutoRun.sln and use interface to interact with LCscaling analysis. This runs UnifiedHydro.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 UnifiedHydro.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.

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%.

Conditions for TC+IC != 100%

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

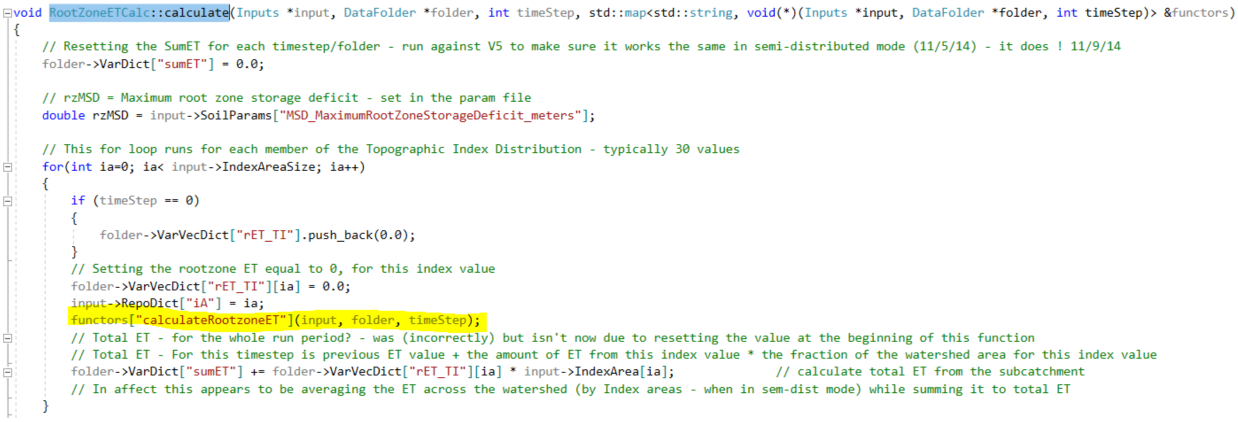
In the Spatial Temperature model, land cover for each dataFolder (a.k.a. cell, pixel in the area of interest raster grid) is defined by a raster layer for tree cover % (treecover.dat), impervious cover % (imperviousCover.dat), and NLCD Land Cover Class (landcover.dat). NLCD Land Cover (LC) classes are 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 UnifiedHydro'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

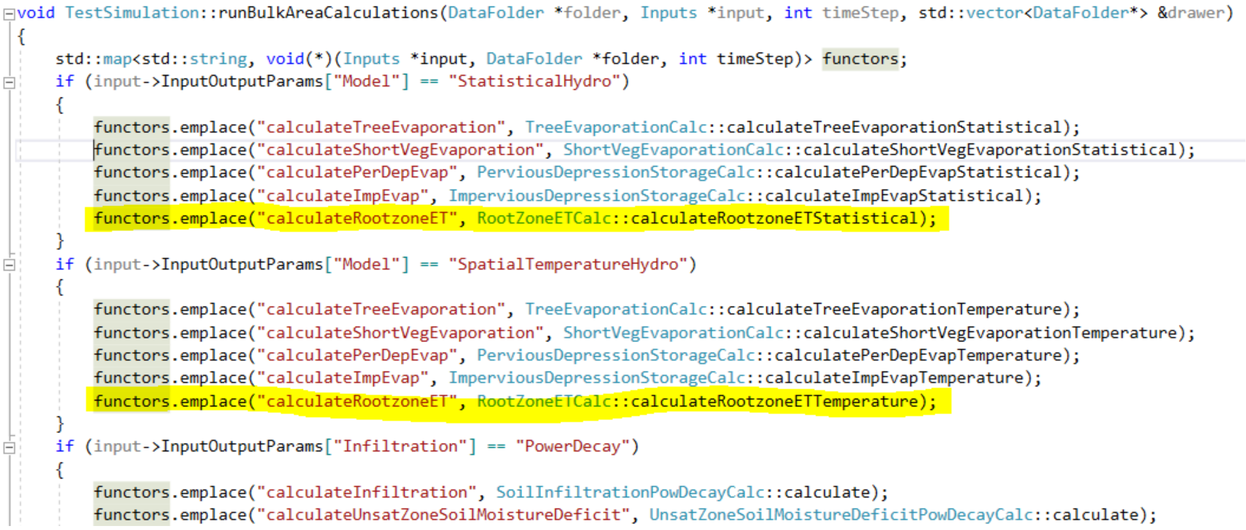
Functors are used instead of complicated switches to plug in certain functions depending on a given condition. For example, whether the semi-distributed hydrology model is run or the spatially-distributed temperature model is run, most of the same functions will be called but certain processes will require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

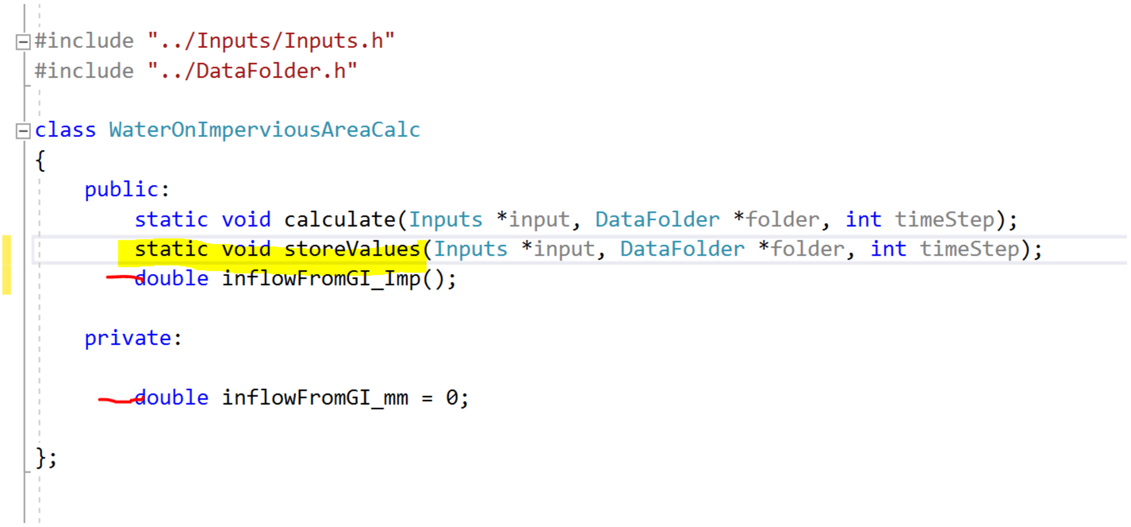
In August 2020 it was observed that the Visual Studio setting for Whole Program Optimization was resulting in build failures for some developers. Whole Program Optimization is disabled moving forward for two reasons: 1) disabling that feature resolves the build errors some developers faced, and 2) this 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.

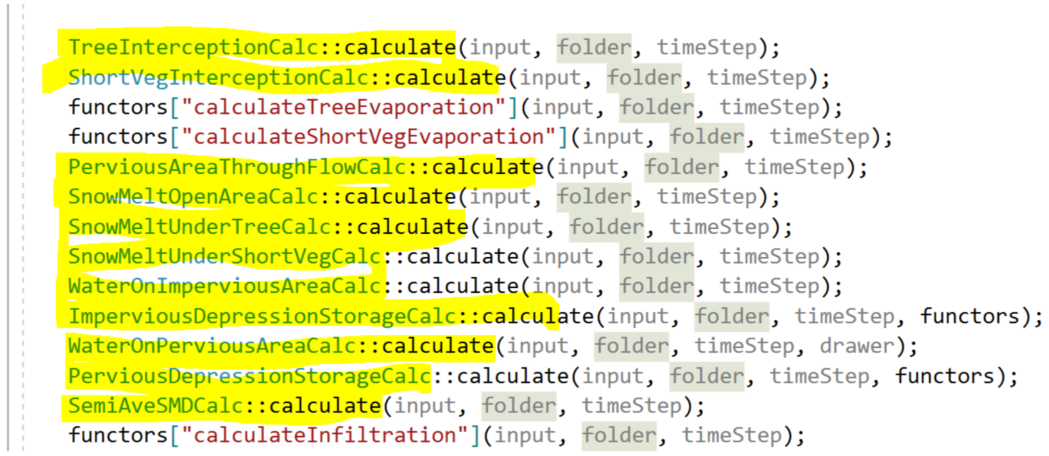
Calculation Classes

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.



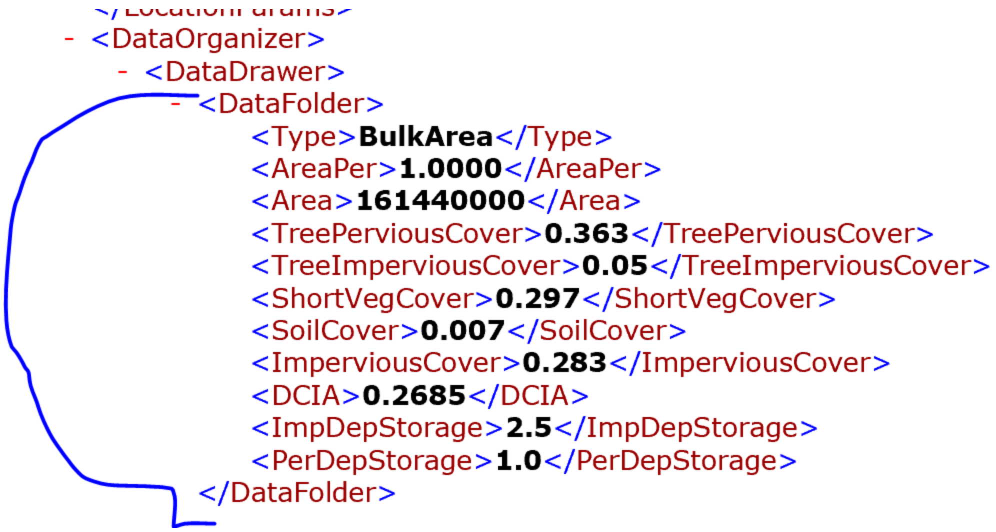
```
TreeInterceptionCalc::calculate(input, folder, timeStep);
ShortVegInterceptionCalc::calculate(input, folder, timeStep);
functors["calculateTreeEvaporation"](input, folder, timeStep);
functors["calculateShortVegEvaporation"](input, folder, timeStep);
PerviousAreaThroughFlowCalc::calculate(input, folder, 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

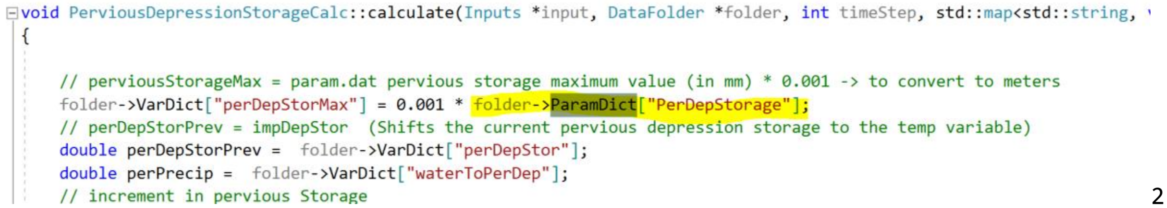
The DataFolder class

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>
```

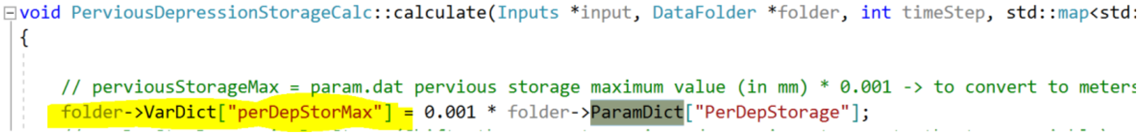


```
void PerviousDepressionStorageCalc::calculate(Inputs *input, DataFolder *folder, int timeStep, std::map<std::string,
{

    // 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.

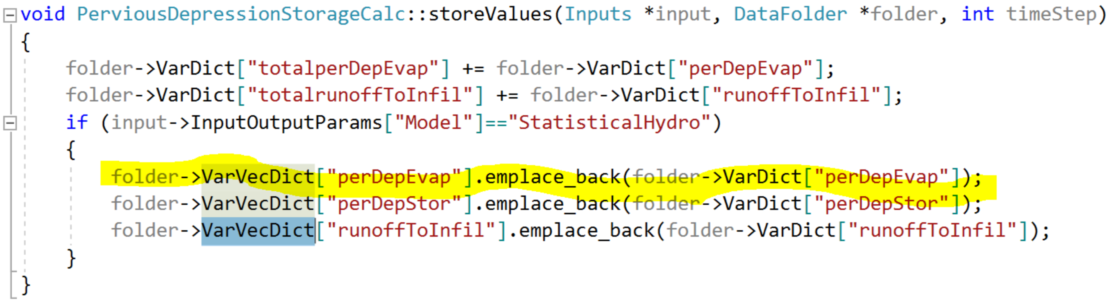
3. This is used to store data created/used by calculations. As opposed to creating class data member. All data specific to a unit such as BulkArea should be stored here. You can create a new entry/start using it without explicitly declaring/defining it. By default, all values will be zero.

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



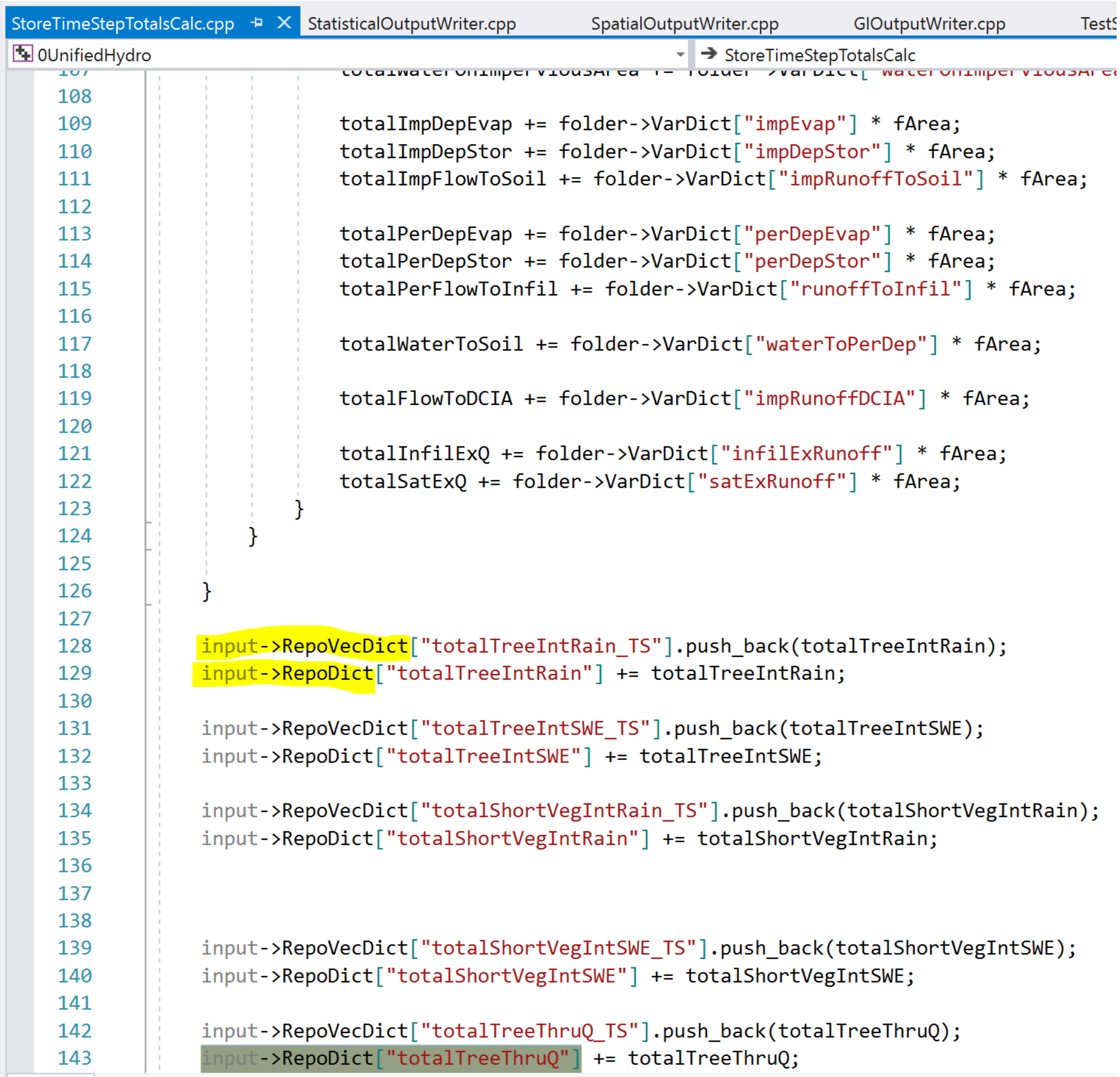
```
void PerviousDepressionStorageCalc::calculate(Inputs *input, DataFolder *folder, int timeStep, std::map<std
{

    // perviousStorageMax = param.dat pervious storage maximum value (in mm) * 0.001 -> to convert to meter
    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.)



```

                totalImpDepEvap += folder->VarDict["impEvap"] * fArea;
                totalImpDepStor += folder->VarDict["impDepStor"] * fArea;
                totalImpFlowToSoil += folder->VarDict["impRunoffToSoil"] * fArea;

                totalPerDepEvap += folder->VarDict["perDepEvap"] * fArea;
                totalPerDepStor += folder->VarDict["perDepStor"] * fArea;
                totalPerFlowToInfil += folder->VarDict["runoffToInfil"] * fArea;

                totalWaterToSoil += folder->VarDict["waterToPerDep"] * fArea;

                totalFlowToDCIA += folder->VarDict["impRunoffDCIA"] * fArea;

                totalInfilExQ += folder->VarDict["infilExRunoff"] * fArea;
                totalSatExQ += folder->VarDict["satExRunoff"] * fArea;
            }
        }

    }

    input->RepoVecDict["totalTreeIntRain_TS"].push_back(totalTreeIntRain);
    input->RepoDict["totalTreeIntRain"] += totalTreeIntRain;

    input->RepoVecDict["totalTreeIntSWE_TS"].push_back(totalTreeIntSWE);
    input->RepoDict["totalTreeIntSWE"] += totalTreeIntSWE;

    input->RepoVecDict["totalShortVegIntRain_TS"].push_back(totalShortVegIntRain);
    input->RepoDict["totalShortVegIntRain"] += totalShortVegIntRain;



    input->RepoVecDict["totalShortVegIntSWE_TS"].push_back(totalShortVegIntSWE);
    input->RepoDict["totalShortVegIntSWE"] += totalShortVegIntSWE;

    input->RepoVecDict["totalTreeThruQ_TS"].push_back(totalTreeThruQ);
    input->RepoDict["totalTreeThruQ"] += totalTreeThruQ;
```

Development method and troubleshooting at revision 279

If a large amount of code has been added and is not compiling, strongly consider starting over from stable checkpoint. First create functions that print the desired inputs to the console. Program a one 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 can not compile, you can not debug. Even if the code compiles, this does not mean it is behaving as expected.

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

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

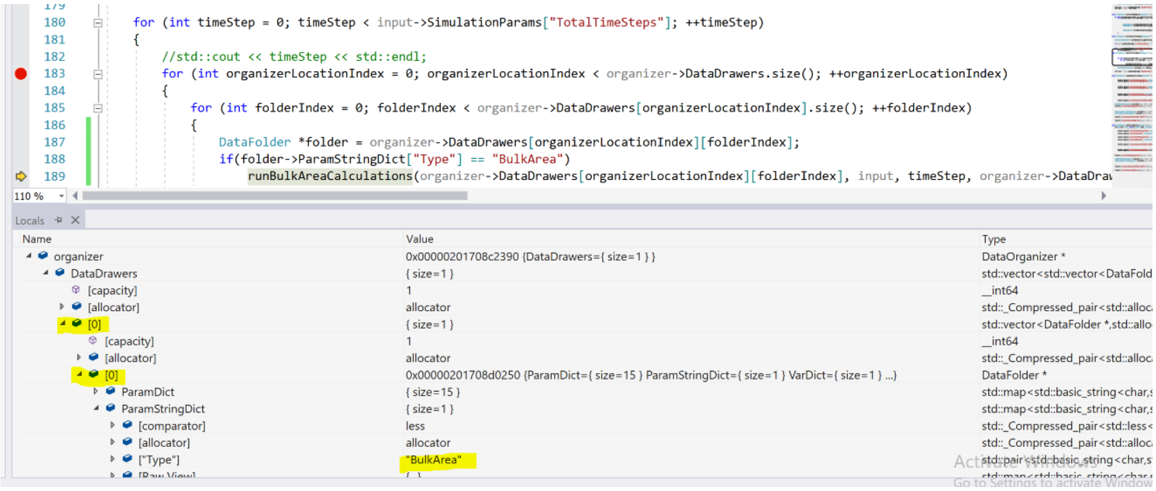
1. Where in the code do we begin running GI calcs? A potentially simple answer: in TestSimulation::RunStatisticalModel, after calling runBulkAreaCalculations, we call runGICalculations. The rationale for having a separate set of calculations to run for bulk area vs. GI is that, though most of the calcs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

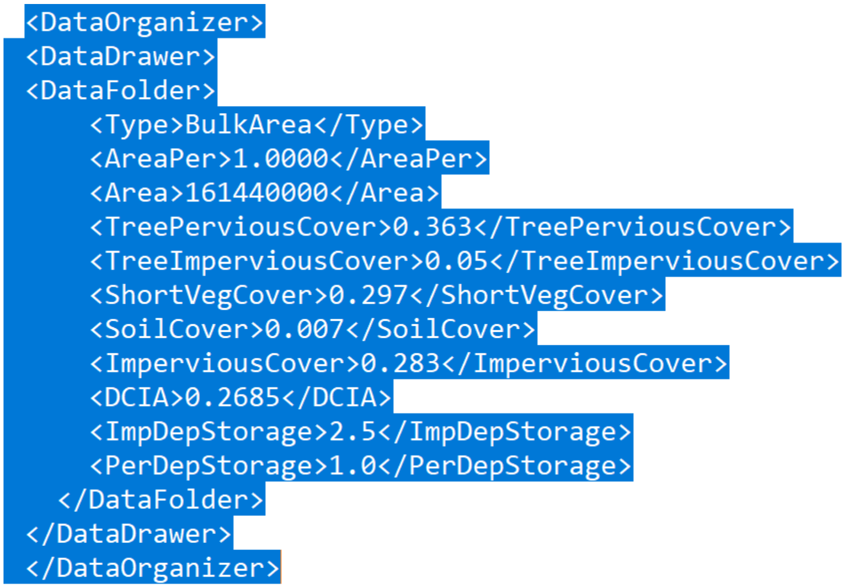
2) What tells the code to grab inputs from the GI structure in the config file rather than bulk area, and how does the code keep track of which structure it is running? For example, bulk area vs. rain garden vs. rain barrel. We can start with just bulk area & rain garden only, but we should keep in mind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



```
for (int timeStep = 0; timeStep < input->SimulationParams["TotalTimeSteps"]; ++timeStep)
{
    //std::cout << timeStep << std::endl;
    for (int organizerLocationIndex = 0; organizerLocationIndex < organizer->DataDrawers.size(); ++organizerLocationIndex)
    {
        for (int folderIndex = 0; folderIndex < organizer->DataDrawers[organizerLocationIndex].size(); ++folderIndex)
        {
            DataFolder *folder = organizer->DataDrawers[organizerLocationIndex][folderIndex];
            if(folder->ParamStringDict["Type"] == "BulkArea")
                runBulkAreaCalculations(organizer->DataDrawers[organizerLocationIndex][folderIndex], input, timeStep, organizer->DataDraw
```

```
Name                        Value                                                                              Type
organizer                   0x00000201708c2390 {DataDrawers={ size=1 } }                                       DataOrganizer *
  DataDrawers               { size=1 }                                                                         std::vector<std::vector<DataFold
    [capacity]              1                                                                                  __int64
    [allocator]             allocator                                                                          std::_Compressed_pair<std::alloc
    [0]                     { size=1 }                                                                         std::vector<DataFolder *,std::allo
      [capacity]            1                                                                                  __int64
      [allocator]           allocator                                                                          std::_Compressed_pair<std::alloc
      [0]                   0x00000201708d0250 {ParamDict={ size=15 } ParamStringDict={ size=1 } VarDict={ size=1 } ...}   DataFolder *
        ParamDict           { size=15 }                                                                        std::map<std::basic_string<char,s
        ParamStringDict     { size=1 }                                                                         std::map<std::basic_string<char,s
          [comparator]      less                                                                               std::_Compressed_pair<std::less<
          [allocator]       allocator                                                                          std::_Compressed_pair<std::alloc
          ["Type"]          "BulkArea"                                                                         std::pair<std::basic_string<char,s
```



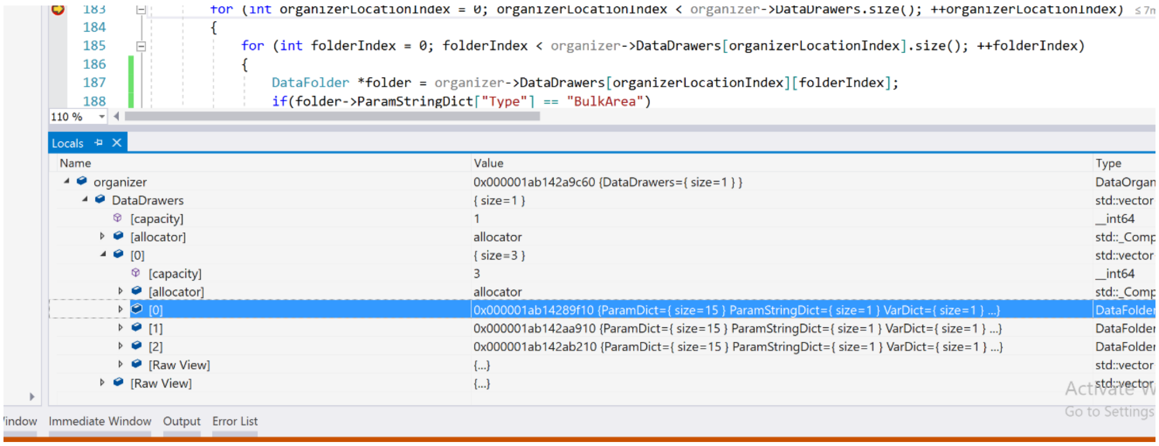
```
<DataOrganizer>
<DataDrawer>
<DataFolder>
     <Type>BulkArea</Type>
     <AreaPer>1.0000</AreaPer>
     <Area>161440000</Area>
     <TreePerviousCover>0.363</TreePerviousCover>
     <TreeImperviousCover>0.05</TreeImperviousCover>
     <ShortVegCover>0.297</ShortVegCover>
     <SoilCover>0.007</SoilCover>
     <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]`

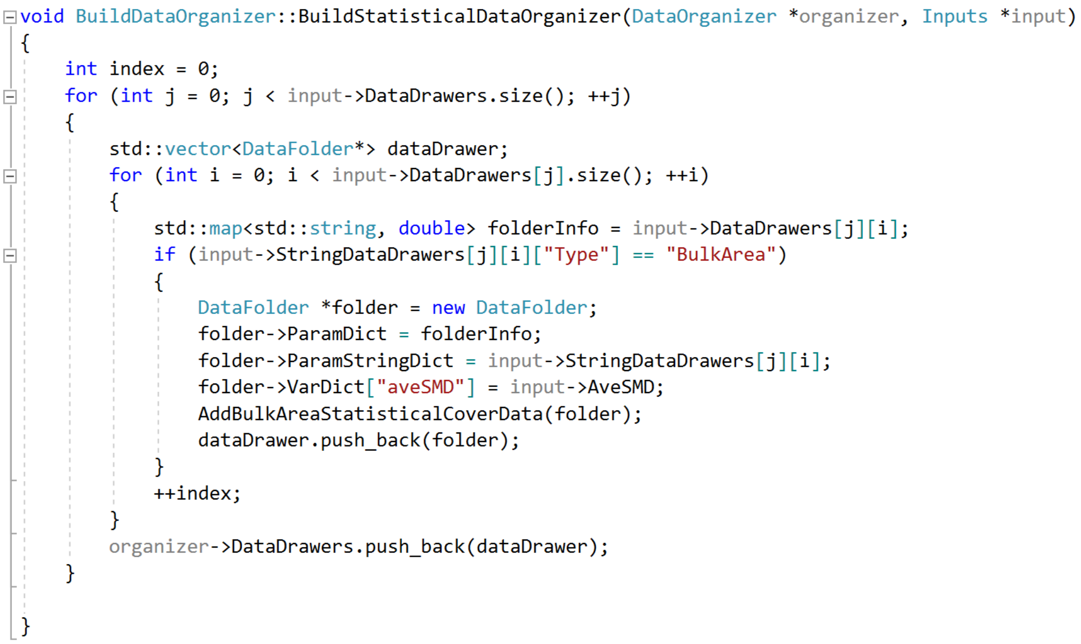


```
for (int organizerLocationIndex = 0; organizerLocationIndex < organizer->DataDrawers.size(); ++organizerLocationIndex)
{
    for (int folderIndex = 0; folderIndex < organizer->DataDrawers[organizerLocationIndex].size(); ++folderIndex)
    {
        DataFolder *folder = organizer->DataDrawers[organizerLocationIndex][folderIndex];
        if(folder->ParamStringDict["Type"] == "BulkArea")
```

```
Locals
Name                 Value                                                                              Type
organizer            0x000001ab142a9c60 {DataDrawers={ size=1 } }                                      DataOrgan
  DataDrawers        { size=1 }                                                                         std::vector
    [capacity]       1                                                                                  __int64
    [allocator]      allocator                                                                          std::_Comp
    [0]              { size=3 }                                                                         std::vector
      [capacity]     3                                                                                  __int64
      [allocator]    allocator                                                                          std::_Comp
      [0]            0x000001ab14289f10 {ParamDict={ size=15 } ParamStringDict={ size=1 } VarDict={ size=1 } ...}   DataFolder
      [1]            0x000001ab142aa910 {ParamDict={ size=15 } ParamStringDict={ size=1 } VarDict={ size=1 } ...}   DataFolder
      [2]            0x000001ab142ab210 {ParamDict={ size=15 } ParamStringDict={ size=1 } VarDict={ size=1 } ...}   DataFolder
      [Raw View]     {...}                                                                              std::vector
    [Raw View]       {...}                                                                              std::vector
```

```
<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.