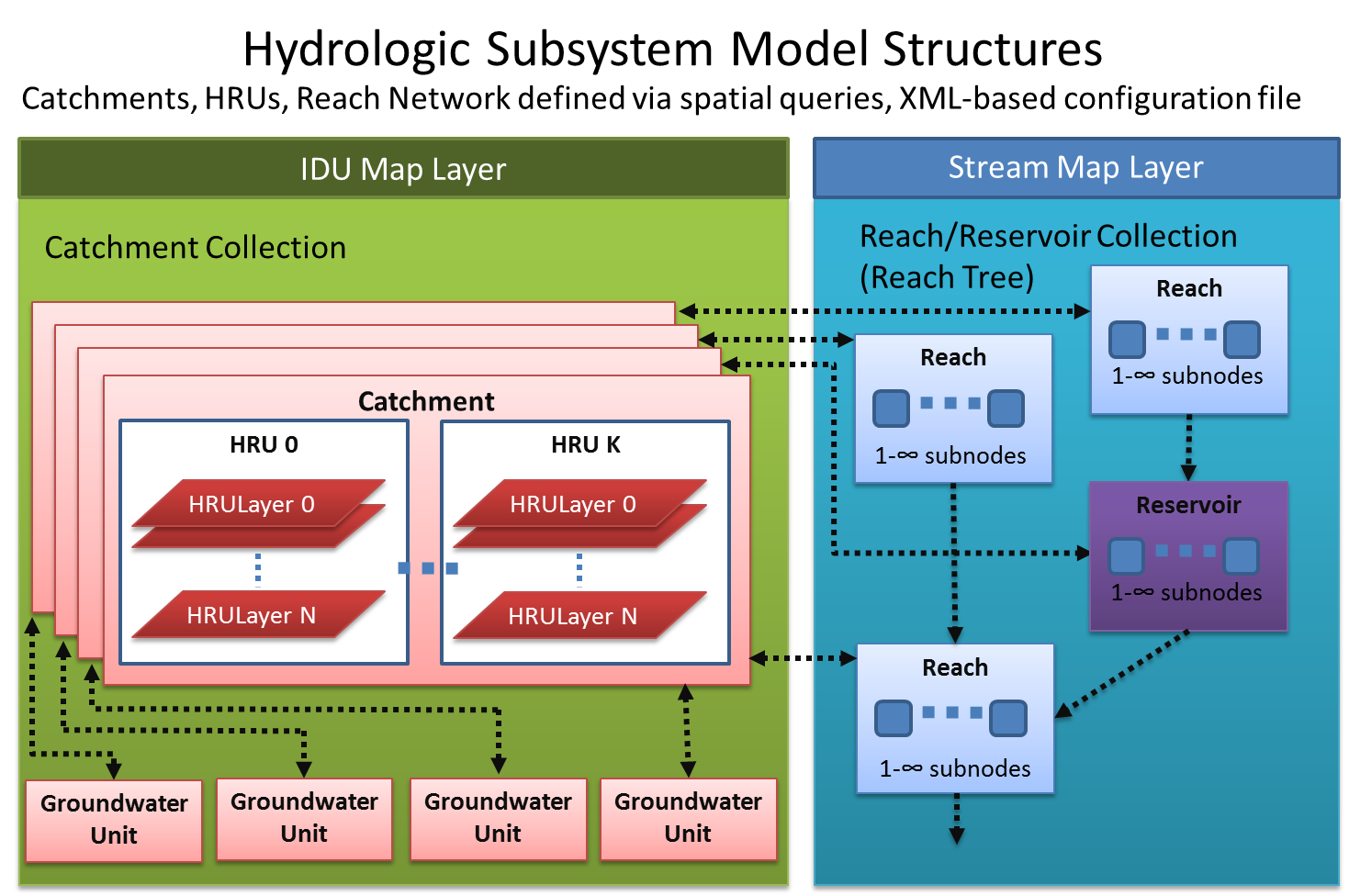
## Flow (Autonomous Process)

**Flow** is a Envision Extension that provides a very flexible framework for hydrologic modeling. Similar to Envision, is supports the idea of “plug-ins”, DLL’s that extend functionality. **Flow** provides a number of elements of hydrological process representation, including geometric representation of terrestrial and aquatic (riverine) datasets used in the hydrologic model, topology, simulation control, data management, and default implementations of important hydrologic processes. While providing for basic hydrologic connectivity (described in detail below), the framework supports the concept of externally defined fluxes for representing sources or sinks at various points in the hydrologic system, i.e. “straws” that the can add, remove, or transfer water at specific locations. In addition, if the various internal implementations of hydrologic processes are insufficient for a given application, the can be globally or locally overridden. Input is an XML file specifying definitions of catchments, hydrologic response units (HRU), and river/stream features, externally-define fluxes and process overrides, and other information needed to define a specific hydrological model. These various framework elements are defined in detail below.

**Spatial Representation**. **Flow** defines (in very flexible ways) a spatial geometry and set of geometric objects that represent spatial elements of hydrologic system (see figure below). Terrestrial elements are organized as Hydrologic Response Units (HRUs) that are represented by collections of polygons in a source layer (typically the IDU layer) with common attributes. HRUs are a fundamental unit of terrestrial hydrologic representation in **Flow**. They can consist of one or more layers (*HRULayer* objects) that maintain water content information and any number of additional state variables (e.g. heat, nitrogen level.) HRU Layers come in three flavors: snow, vegetation, and soil; any number (including zero) of any of these layer types can be defined, but they are always sequenced top to bottom with snow on top, vegetation in the middle and soil layers on the bottom. HRUs are collected into catchments (*Catchment* objects) with a single drainage point. These are assembled by **Flow** from a GIS layer (*MapLayer* object) managed by Envision, typically the IDU layer, that has relevant spatial data used to define or model the hydrologic system. Similarly, the stream/river network is represented by a linear feature network coverage (*MapLayer* object), also managed by Envision, that represents the river system network layer that resides “on top” of the IDU coverage. Reaches are subdivided into “subnodes” that can be spaced as needed for a particular model construct. *Catchments*, *HRUs* and *Reaches* are defined by “spatial aggregates”, collections of polygons that have similar attributes. *HRULayers* always nest within an *HRU*, HRU’s should always nest in a catchment. The spatial domain of these aggregates can be limited, meaning a subset of base layer terrestrial polygons or stream layer lines can by selected into the aggregates; any excluded polygons/lines are simply ignored by the model.

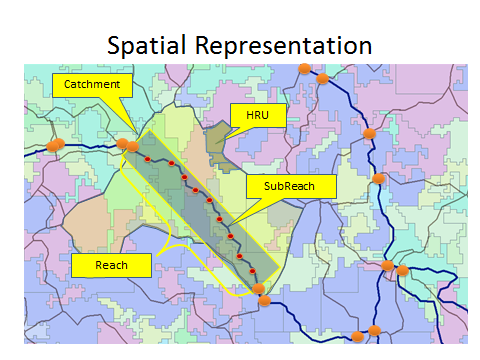
State variable information is maintained at the *HRULayer* level and at the Reach subnode level. By default, a single state variable related to water content is maintained by these containers. Additional state variables, representing transported constituents, can be easily added by specifying them in the input XML file.

Topologically, **Flow** defines and manages hydrological connectivity between *Catchments* and *Reaches* – catchments “know” what reach they are connected to, and vice versa. **Flow** assumes that terrestrial/riverine connectivity is between catchments and reaches only; i.e. all lateral inflow/outflow from a catchment’s *HRULayers* is collected at the catchment level and transferred to the reach corresponding to a catchment. Connectivity is defined through a join column that must be present in the catchment coverage, with a related join column in the reach coverage; catchments are joined to reaches that share common values in their join fields; this is assumed to be a 1:1 relationship; one catchment can be associated with no more than one reach, and one reach can be associated with no more than one catchment.



**Internal Hydrologic Processes**. **Flow** provides several internal representations of important hydrologic processes, including vertical transport among HRULayers, transfer of water/constituents between catchments and reaches, and instream routing. Developers can specify which internal method to use, or override them completely and provide an externally defined method. In this latter case, a DLL and function entry point must be specified (a “global handler”), and appropriate algorithms provided in that entry point. This external function has full access to the internal representations provided by flow. Details are providing in the “Writing a Global Handler” section below.

**Defining External Fluxes**. Central to **Flow** is the idea that external fluxes (“straws”) can be added to hydrological models easily and flexibly. Fluxes can represent wells, surface water extraction/additions, precipitation, processes like ET, or any other hydrology-related flux. Fluxes can be defined via an external plugin, an external data source, or, in simple cases, by specification of evaluable expression in the **Flow** input file. Fluxes can be “sources” or “sinks”, and can be one-way or two-way straws. The straw “ends” can be inserted into a *HRULayer,* a *Reservoir*, a *Groundwater unit* or a *Reach*.



The **Flow** process and individual **Flow** elements are defined in an Xml input file that is specified in the Project (.ENVX) file. An example input files is given below. .

To include a Flow-based mode in an application, include the **<autonomous\_process>** element shown below in the **<autonomous\_process>** section of the Project file. Note that the **Flow** Xml input file is given as the initInfo. This Flow entry corresponds to the Flow file given above. Note that the specified file must be either in the ENVISION executable directory, or specified as a fully-qualified path.

<autonomous\_processes>

<autonomous\_process

name ='My Hydro Model'

path ='flow.dll'

id ='0'

use ='1'

timing ='0'

freq ='1'

sandbox ='0'

fieldName =''

initInfo ='flow\_input.xml'

/>

</autonomous\_processes>

The format of the Xml input file (named in the initiInfo field above) consists of 7 key elements, each defined with a particular tag. The <flow\_model> tag defines an element with attributes that outline the highest level of Flow requirements.

<!-- <flow\_model> defines a model

Attributes include:

name: Name of the model. (optional)

time\_step: Time step at which all modeled components are synchronized. This should be at least as large as the largest component time step specfified in the component tags below. (required)

init\_catchment\_cols: flag indicating whether Flow should build catchments from the aggregate queries defined in the <catchment> tag below and populate the catchmentID\_col with unique catchment IDs, or assume this information is already in the catchment coverage. 1=build catchments, 0=read from coverage. (optional, default is "1" )

join\_col: catchment coverage column name containing the join information for connecting catchments and reaches. (required, typically COMID)

-->

<flow\_model name="wwflow" time\_step="1" init\_catchment\_cols="1" join\_col="COMID">

The <catchment> tag defines an element used to outline how data will be utilized to define the upslope model

<!-- <catchment> specifies how to construct catchment structures for the model.

Attributes include:

layer: Name of the layer used to construct catchments from (in the envx file). This is generally the IDU layer. (optional, defaults to first layer)

query: Subset of the layer used to create catchments. If empty or not present, the entire layer is used. (optional)

area\_col: column name containing the area of the polygon. (optional, defaults to "AREA")

catchmentID\_col: column name containing the catchmentID identifying unique catchments, generated during the aggregation process. This column will be added to the layer if it doesn't already exist. (optional - defaults to CATCH\_ID)

hruID\_col: column name containing the hruID identifying unique catchments, generated during the aggregation process. This column will be added to the layer if it doesn't already exist. (optional - defaults to HRU\_ID)

catch\_agg\_cols: Comma-separated list of column names used to identify unique combinations of attribute values during the catchment aggregation process. (required)

hru\_agg\_cols: Comma-separated list of column names used to identify unique combinations of attribute values during the hru aggregation process. (required)

soil\_layers: integer identifying number of HRU soil layers to use (optional, defaults to 1)

snow\_layers: integer identifying number of HRU snow layers to use (optional, defaults to 0)

veg\_layers: integer identifying number of HRU vegetation layers to use (optional, defaults to 0)

layer\_names: colon-separated list of layer names used to describe HRU layers for plotting purposes. the sum of soil\_layers +snow\_layers +veg\_layers should be equal to the number of names (optional)

init\_water\_content: initial soil water content (m3/m3) - (optional, defaults to 0.5)

-->

<catchment layer="IDU" query="" area\_col="AREA" catchmentID\_col="CATCH\_ID" hruID\_col="HRU\_ID" catchment\_agg\_cols="COMID" hru\_agg\_cols="LULC\_B" soil\_layers="5" snow\_layers="0" veg\_layers="1" layer\_names="Canopy:Impervious:Soil Recharge Zone:Lower Zone:Subsurface Reservoir:Groundwater Reservoir"/>

The <stream> tag is used to define how state variables will be defined from the stream network

<!-- <stream>

Attributes include:

layer: Name of the layer used to construct catchments from (in the envx file). (required)

query: Subset of the layer used to create reaches. If empty or not present, the entire layer is used. (optional)

order\_col: Column name to populate with stream order info. If the column doesn't exist in the stream coverage, it will is generated.)

subnodeLength: max length between subnodes (length units corresponding to the coverage) (optional, defaults to 0, which allocates one subnode per reach.

wd\_ratio: default value for width/depth ratio for reach segments. (optional, defaults to

method: solution method, euler, rk4, rkf, kinematic, external. Note external implies fluxes are processed by the plugin, not the framework. (optional, defaults to 10)

stepsize: timestep used to solve the instream routing equations. (optional, defaults to 1 day)

-->

<stream layer="Streams" query="" subnode\_length="1000" order\_col="ORDER" />

The <global\_methods> tag defines an element outlining solution procedures

<!-- <global\_methods>

Attributes include:

reach\_routing: reach routing solution method, one of 'euler', 'rk4', 'rkf', 'kinematic', or an externally defined method. Externally defined methods should be of the form: 'path:entrypointname'. Note external implies fluxes are processed by the plugin, not the framework. (optional, defaults to "kinematic")

horizontal\_exchange: solution method for lateral exchange between reaches and corresponding HRU Layers; internal options include 'linear\_reservoir'. Externally defined methods should be of the form:'path:entrypointname'. (optional, defaults to "linear\_reservoir")

hru\_vertical\_exchange: solution method for vertical movement of water between HRULayers; internal options include 'brooks\_corey'. Externally defined methods should be of the form:'path:entrypointname'. (optional, defaults to "brooks\_corey")

-->

<global\_methods reach\_routing="kinematic" horizontal\_exchange= "fn=WHydro.dll:PRMS\_Runoff" hru\_vertical\_exchange="fn=WHydro.dll:PRMS\_Vertical" />

The < fluxes> tag defines fluxes that will be used to move water into and out of the system, as well as between different elements (state variables) within the modeled system.

<fluxes>

<!-- <flux> - Specifies all fluxes.

Attributes include:

name: the name of the flux (required)

description: a text description of this flux

path: the path to the DLL that contains the function returning a flux, OR a path to a datasource. This is required unless a "value" attribute is specified; these are mutually exclusive attributes.

flux\_handler: the 'C' function name entry point in the DLL - valid only for externally defined functions

type: reach, hruLayer, reservoir

query: where does this flux apply

source: fn=<dllpath:function> for functional, db=<datasourcepath:columnname> for datasets

-->

<flux name="Precipitation" path="WHydro.dll" flux\_handler="PrecipFluxHandler" description="" source\_type="hrulayer:0" source\_query="AREA > 0" />

<flux name="Evaporation from Vegetation" path="WHydro.dll" description="" sink\_type="hrulayer:0" sink\_query="AREA > 0" flux\_handler="ETFluxHandler" />

The < tables> tag defines fluxes that will be used to move water into and out of the system, as well as

between different elements (state variables) within the modeled system.

<tables>

<!--

specifies table inputs.

name: name for table, this MUST correspond to a field in the IDU coverage

source: path to csv file containing values and label headers

type: 'float' - all values are real, 'int' - all values are integer, 'var' - variable type (only float currently supported)

-->

<table name="LULC\_A" description="soil hydrologic group" col="LULC\_A" source="c:\envision\StudyAreas\WW2100\prms\_lulc.csv" type="float" />

<table name="GEO" description="geology" col="GEO" source="c:\envision\StudyAreas\WW2100\prms\_geology.csv" type="float" />

<table name="Basin" description="Basinwide calibration parameters" col="SUBBASIN" source="c:\envision\StudyAreas\WW2100\prms\_basin.csv" type="float" />

</tables>

The <sample\_locations> tag defines places in the system for which time series data will be saved. The model will store data for each location representing both reaches and hru layers. Currently, discharge (m3/s)for each reach is collected, along with water content (in m3/m2) for each layer in the first HRU of each catchment defined by the id.

<sample\_locations>

<!--

name: name for the location

description: more complete description

col: name of database column used to represent locations

id: value in the database column representing the location of interest

-->

<reach\_location name='Outlet' description="Reach with the largest contributing area" col="COMID" id="23764545" />

<reach\_location name='Mainstem' description=" " col="COMID" id="23763585" />

**Defining the default methods**.

The model includes default methods to move water between upland HRUlayers and the reach network. These default methods are always available to users, although it is anticipated that they will frequently be overridden by more project specific models.

**Instream routing: a kinematic approach**

The default method for routing instream flows is a kinematic approach that follows directly from a solution developed by Chow et al. (1988). The method is a simplification of the St. Venant equations describing free surface flows. The full St Venant represent the conservation of mass and momentum and can be written as:

Continuity



Conservation of Momentum



Local acceleration

Acceleration

Convective acceleration

Acceleration

Pressure Force

Gravity Force

Friction Force

These two equations fully describe discharge as a function of both time and space, and a direct solution to these equations is typically referred to as a dynamic solution. Simplifications are commonly employed however, and rely upon a variety of assumptions about the movement of water and the importance of different terms. The kinematic wave is one such simplification. If the kinematic wave dominates the hydrograph, the acceleration and pressure force are negligible and the model simplifies to the statement that gravity forces equal friction forces. In other words, within each element of time and space the discharge does not change, a situation commonly referred to as uniform flow. In the event that the flow can be considered uniform, the momentum equation can be replaced with a standard uniform resistance equation such as Manning’s or the Chezy equation. A commonly cited version of the kinematic flow equation, using Manning’s equation, is:



In this equation, alpha and beta are parameter combinations from Manning’s equation.

A finite difference solution the kinematic wave solution can be developed by assuming



which when substituted into 1 can solved for the the unknown value Qtime . Assuming that

the finite difference solution becomes



This equation represents the unknown value of Q as a function of known values related to upstream discharges, previous time discharge, and normal flow parameters and was developed by (Chow et al., 1998). The tree representation provided by Envision creates the topology necessary to solve the equation, and it is important to recognize that users will likely override the default model, in particular for instances where the kinematic assumptions are violated.

**Horizontal Exchanges: a linear reservoir approach**

The default method to define exchange between HRULayers and connected stream reaches is a very simple linear reservoir. It assumes that this exchange occurs between the bottommost HRULayer and the stream and that the rate is defined as:



Where he is the horizontal flux rate (m/d) k is a linear coefficient (1/d) and theta is the water content in the lowest HRULayer (m3/m2). The total flux into the stream reach at any point in time is the sum of the horizontal exchanges from each HRU in the catchment.

**Vertical Exchanges: a Brooks Corey approach**

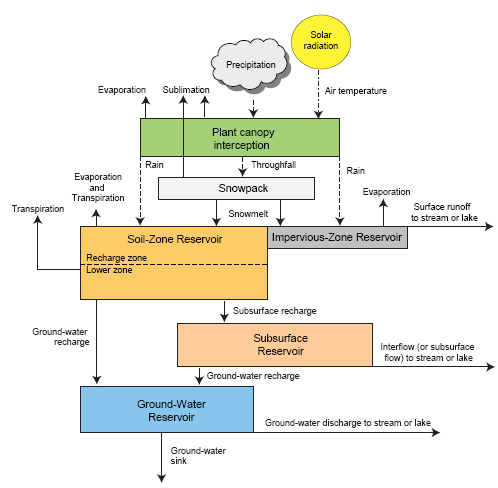
Vertical exchanges are characterized in the default methods by a Brooks Corey type assumption where:



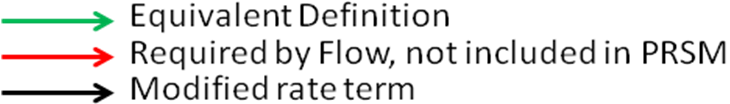
Where K(is the flux from a layer to the layer just below it (m/d), is the volumetric water content, s is the residual water content, r is the wilting point and  is the pore size distribution index.

**Defining externally derived methods**. Flow allows for the definition of any number of user defined methods to capture the dynamics defining stocks and flows. These methods can be used to override the default methods provided by Flow. Overrides are useful when there is a particular conceptual model of catchment hydrology that the user would like to employ, and represent a fundamental component of the framework. The development of a model override requires the user to develop a plugin to Flow, which is cast as a dll.

As an example of what a Flow override might look like, we have developed a plug-in where the vertical and horizontal fluxes for elements representing vegetation, soils, and groundwater are based upon those defined by the PRMS model (Leavesley et al., 1983). While the close correspondence between the elements suggests that parameterizations derived using the original version PRMS may be very useful in developing calibrated simulations in Flow-PRMS, the two will not produce equivalent results. This is because many elements (meteorology, snow accumulation and melt, etc) are entirely different and because of the differences imposed upon Flow-PRMS (outlined in the figure below) to insure compatibility with the framework.



A conceptual diagram taken from the original PRMS documentation (Leavesley et al., 1988



A conceptual diagram outlining elements of PRMS that have been incorporated into the Flow-PRMS plugin. In some cases, rate terms had to be modified or added to produce a version of the conceptual model that was conformant with FLOW.

The PRMS override is incorporated into Flow through the XML specification, with the primary element being the <global method> tag

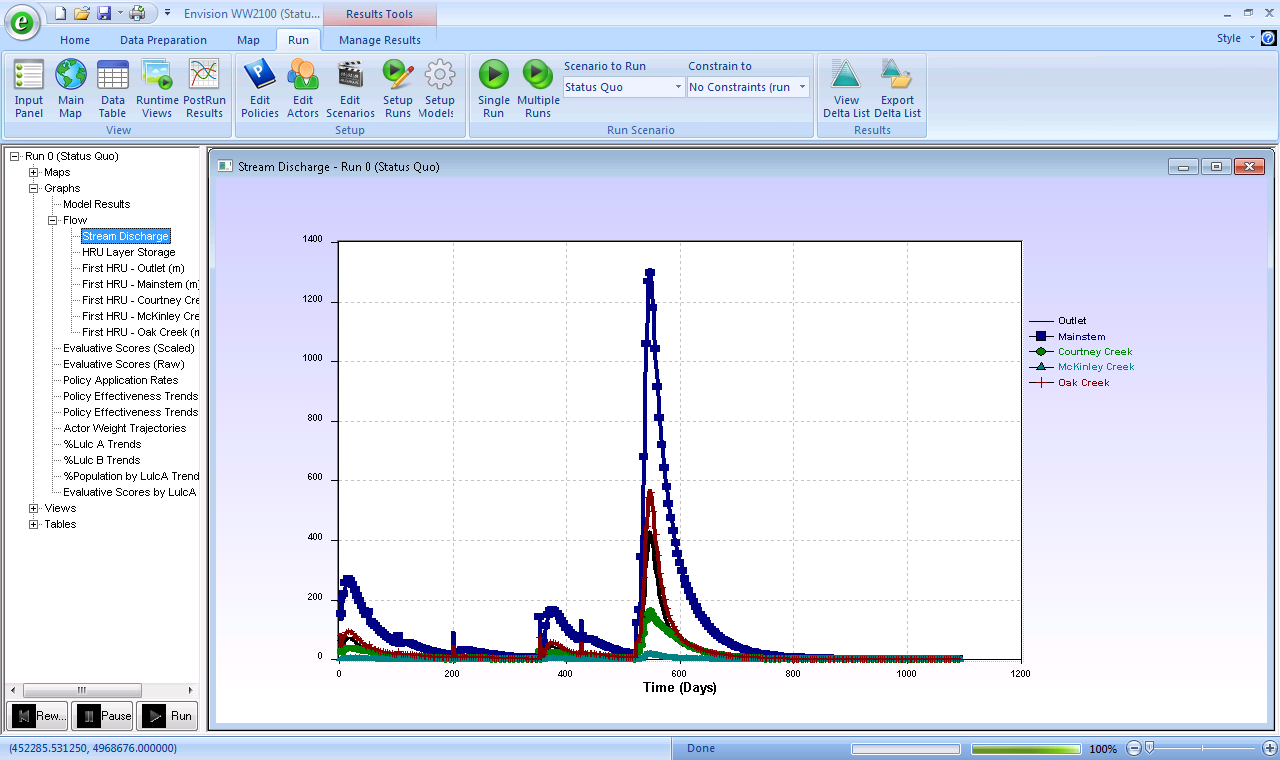
<global\_methods reach\_routing="kinematic" horizontal\_exchange="fn=WHydro.dll:PRMS\_Runoff" hru\_vertical\_exchange="fn=WHydro.dll:PRMS\_Vertical" />

This element indicates that the reach routing is to be captured using the framework supported default model, but that the horizontal and vertical exchanges are to be overridden from within a dll called WHydro.dll using the methods PRMS\_Runoff and PRMS\_Vertical respectively. Both of these methods must be exported from the WHydro.dll (the plugin). Each method returns a different exchange value depending upon the particular layer, the value of state, as well as a set of parameters that depend upon soils, geology, and basin-specific calibrations. For reference, the C++ implementation for these methods has been included in appendix A. The exploratory code presented here is under current development and changes frequently.

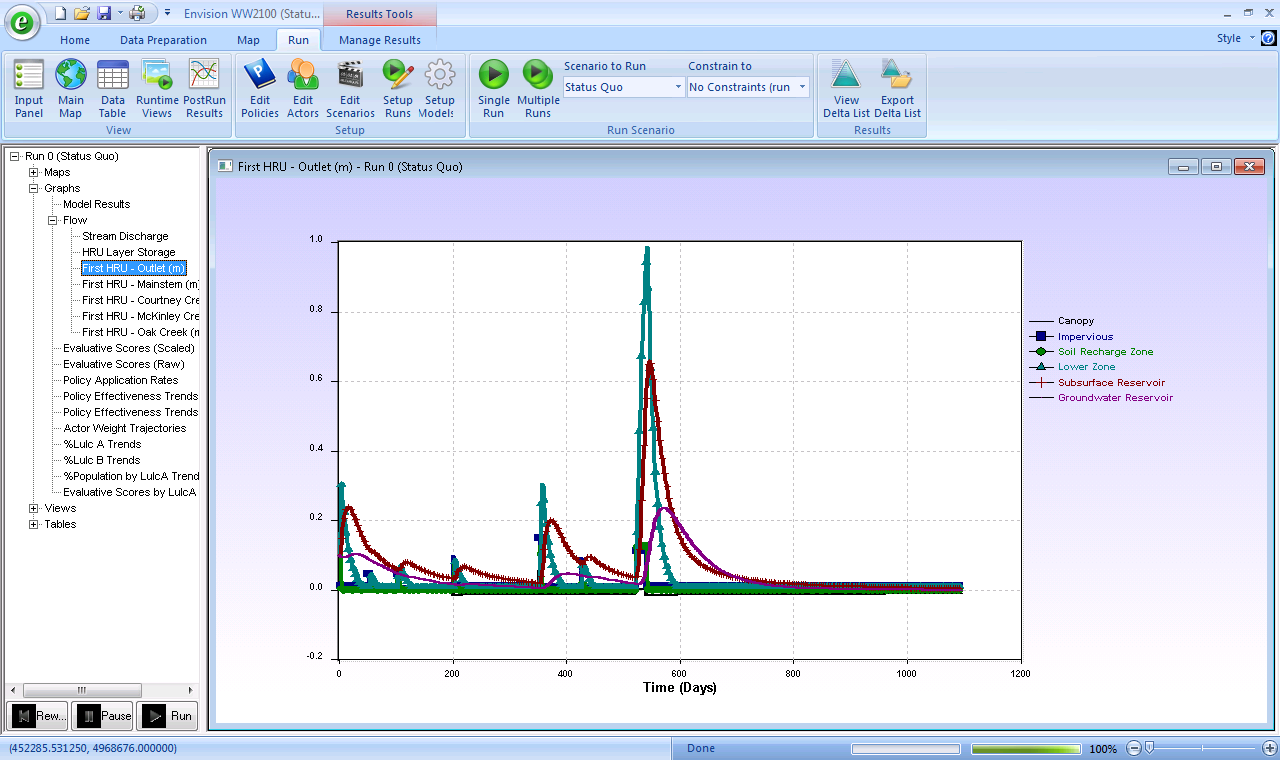
These exchange terms comprising the override have been defined as follows

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| HRU  Layer | State Variable Name | Exchange Term | Type | Definition | Category |  |
| 0 | Plant Canopy | Throughfall | HRULayer Exchange | PTF=PPT-(STOR-XIN) when PPT > (STOR-XIN)  PTF=0 when PPT < (STOR-XIN) | Equivalent |
| 1 | Impervious | ImperviousToRecharge | HRULayer Exchange | SRO=RSTOR-RETIP when RSTOR > RETIP  SRO = when RSTOR < RETIP | Similar Rate |
| 2 | Recharge Zone | RechargeToLower SoilZone | HRULayer Exchange | Rec\*k1 | New Rate |
| 3 | Lower Soil Zone | SubsurfaceRecharge | HRULayer Exchange | SRO = CAP\*PTN | Equivalent |
| 4 | Subsurface Reservoir | GroundwaterRecharge | HRULayer Exchange | GAD=RESP\*(RES/RESMX)^REXP | Equivalent |
| 5 | Groundwater Reservoir | GroundwaterLoss | HRULayer Exchange | SNK=GSNK\*GW | Equivalent |
|  |  |  |  |  |  |
| 1 | Impervious | SurfaceRunoff | HRU Reach Exchange | out = (waterDepth-SRO)\*k2 | Similar Rate |
| 4 | Subsurface Reservoir | Interflow | HRU Reach Exchange | RAS=RCF\*RES+RCP\*RES^2 | Equivalent |
| 5 | Groundwater Reservoir | GroundwaterToStream | HRU Reach Exchange | BAS=RCB\*GW | Equivalent |

Key rate terms that are similar between the original definition of PRMS and the PRMS-based plugin to Flow. Terms that are described in capital letters have been taken directly from PRMS, while those with lower case letters have been added to the model to ensure Flow conformance. In this case, added terms were necessary because of the Flow specification that each HRULayer define the rate at which water drains from it. The original PRMS does not include explicit definitions of vertical flows from the impervious zone and soil zone reservoirs, but instead employs a bypass mechanism, when those elements are at capacity, to move water through the HRU Layers. A more complete definition of the model terms can be found in Leavesley et al.(1983). A typical set of outputs for this model, and for a given parameterization are outlined below.



An Envision-based figure outlining the simulated stream discharge for each selected reach in a prototype model based upon the Calapooia River. The model is based upon synthetic precipitation data and comparisons to measured discharge have not been developed



An Envision-based figure outlining the HRULayer state variables for a single reach and HRU within that catchment. These dynamics, in concert with similar patterns across all HRUs, result in the discharge dynamics outlined in the figure above.

References

Chow. V., Maidment, D., Mays L., 1988. Applied Hydrology, McGraw-Hill, New York. 572 pp.

Leavesley, G.H., Lichty, R.W., Troutman, B.M., Saindon, L.G., 1983. Precipitation–Runoff Modeling System: User’s Manual, US Geological Survey Water Resources Investigation Report 83-4238 p. 207..

Appendix A. Implementation for the Override plugin. A complete version of the current code can be accessed directly through an available subversion site.

// WHydro.cpp : Defines the DLL.

//

#include "stdafx.h"

#include "WHydro.h"

#include <Flow\Flow.h>

#include "GDALWrapper.h"

#include "GeoSpatialDataObj.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

float WHydro::PRMS\_Vertical( FlowContext \*pFlowContext )

{

int catchmentCount = pFlowContext->pFlowModel->GetCatchmentCount();

GeoSpatialDataObj \*myGeo;

CString filein = "\\envision\\StudyAreas\\WW2100\\gridded\_obs\_daily\_Tavg\_1999.nc";//geographic coordinates...need to reproject

CString varin = "Tavg";

myGeo->ReadSpatialData(filein,varin,0);

ParamTable \*pLULC\_Table = pFlowContext->pFlowModel->GetTable( "LULC\_A" ); // store this pointer (and check to make sure it's not NULL)

int col\_imperv = pLULC\_Table->GetFieldCol( "IMPERV" ); //percent impervious area

int col\_covnds = pLULC\_Table->GetFieldCol( "COVNDS" ); //summer cover density for major vegetation for each HRU;

int col\_scx = pLULC\_Table->GetFieldCol( "SCX" ); //percent Maximum possible contributing area for surface runoff as proportion of each HRU

int col\_scn = pLULC\_Table->GetFieldCol( "SCN" ); // inches coefficient in surface runoff contributing area-soil moisture index relation

int col\_retip = pLULC\_Table->GetFieldCol( "RETIP" ); //maximimum retention storage capacity (here in inches)

int col\_sc1 = pLULC\_Table->GetFieldCol( "SC1" ); //Coefficient in surface runoff contributing area—soil moisture index relation

ParamTable \*pGeo\_Table = pFlowContext->pFlowModel->GetTable( "GEO" );

int col\_smax = pGeo\_Table->GetFieldCol( "SMAX" ); //inches maximum available water holding capacity in profile

ParamTable \*pBasin\_Table = pFlowContext->pFlowModel->GetTable( "Basin" );

int col\_rsep = pBasin\_Table->GetFieldCol( "RSEP" ); //daily recharge coefficient

int col\_resmx = pBasin\_Table->GetFieldCol( "RESMX" ); //coefficient

int col\_rexp = pBasin\_Table->GetFieldCol( "REXP" ); //coefficient

int col\_rcp = pBasin\_Table->GetFieldCol( "RCP" ); //nonlinear routing coefficient for each subsurface reservoir See Laenen et al

int col\_rcb = pBasin\_Table->GetFieldCol( "RCB" ); //gw reservoir routing coefficient

int col\_rcf = pBasin\_Table->GetFieldCol( "RCF" ); //linear routing coefficient for each subsurface reservoir

// iterate through catchments/hrus/hrulayers, calling fluxes as needed

for ( int i=0; i < catchmentCount; i++ )

{

Catchment \*pCatchment = pFlowContext->pFlowModel->GetCatchment(i);

float precip = myGeo->Get(pCatchment->m\_centroid.x,pCatchment->m\_centroid.y);

//float precip = pCatchment->m\_currentPrecip;

int hruCount = pCatchment->GetHRUCount();

for ( int h=0; h < hruCount; h++ )

{

HRU \*pHRU = pCatchment->GetHRU( h );

int hruLayerCount=pHRU->GetLayerCount();

float IMPERV,RETIP,COVDNS,SMAX,RCF,RSEP, RESMX,REXP,SC1,SCN,SCX,RCP,RCB=0.0f;

VData lulcA,geo,basin;

pFlowContext->pFlowModel->GetHRUData( pHRU, pLULC\_Table->m\_iduCol, lulcA, DAM\_FIRST ); // get HRU info

if (pLULC\_Table->m\_type==DOT\_FLOAT)

lulcA.ChangeType(TYPE\_FLOAT);

bool ok = pLULC\_Table->Lookup( lulcA, col\_imperv, IMPERV ); // get param value from the table

ok = pLULC\_Table->Lookup( lulcA, col\_covnds, COVDNS ); // get param value from the tabl

ok = pLULC\_Table->Lookup( lulcA, col\_scn, SCN ); // get param value from the table

ok = pLULC\_Table->Lookup( lulcA, col\_scx, SCX ); // get param value from the table

ok = pLULC\_Table->Lookup( lulcA, col\_retip, RETIP ); // get param value from the table

ok = pLULC\_Table->Lookup( lulcA, col\_sc1, SC1 ); // get param value from the table

pFlowContext->pFlowModel->GetHRUData( pHRU, pGeo\_Table->m\_iduCol, geo, DAM\_FIRST ); // get HRU info

if (pGeo\_Table->m\_type==DOT\_FLOAT)

geo.ChangeType(TYPE\_FLOAT);

ok = pGeo\_Table->Lookup( geo, col\_smax, SMAX ); // get param value from the table

pFlowContext->pFlowModel->GetHRUData( pHRU, pBasin\_Table->m\_iduCol, basin, DAM\_FIRST ); // get HRU info

if (pBasin\_Table->m\_type==DOT\_FLOAT)

basin.ChangeType(TYPE\_FLOAT);

ok = pBasin\_Table->Lookup( basin, col\_rsep, RSEP ); // get param value from the table

ok = pBasin\_Table->Lookup( basin, col\_resmx, RESMX ); // get param value from the table

ok = pBasin\_Table->Lookup( basin, col\_rexp, REXP ); // get param value from the table

ok = pBasin\_Table->Lookup( basin, col\_rcp, RCP ); // get param value from the table

ok = pBasin\_Table->Lookup( basin, col\_rcb, RCB ); // get param value from the table

ok = pBasin\_Table->Lookup( basin, col\_rcf, RCF ); // get param value from the table

CString msg;

if ( ! ok )

{

msg.Format("Flow plugin couldn't find PRMS parameter RCP for lulcA %i (looked in column %i)",lulcA, col\_retip);

Report::ErrorMsg(msg);

}

for ( int l=0; l < hruLayerCount; l++ ) //All but the bottom layer

{

HRULayer \*pHRULayer = pHRU->GetLayer( l );

float value = 0; float throughfall=0.0f;

float d=pHRULayer->m\_volumeWater/pHRU->m\_area;

switch( pHRULayer->m\_layer )

{

case 0:

value = PRMS\_Interception(d, precip, throughfall, COVDNS);

break;

case 1:

value = PRMS\_Impervious(d,IMPERV,RETIP);

break;

case 2:

value = PRMS\_RechargeZone(d,IMPERV,SCN,SCX, SC1);

break;

case 3:

value = PRMS\_LowerZone(d,SMAX);

break;

case 4:

value = PRMS\_SubsurfaceReservoir(d,RCP,RCF,RSEP,RESMX,REXP);

break;

case 5:

value = PRMS\_GroundwaterReservoir(d,RCB);

break;

default:

value=0.0f;

} // end of switch

pHRULayer->m\_verticalDrainage = value\*pHRU->m\_area;

pHRULayer->m\_wc = pHRULayer->m\_volumeWater/pHRU->m\_area;

}

}

}

return 0.0f;

}

float WHydro::PRMS\_Runoff( FlowContext \*pFlowContext )

{

int catchmentCount = pFlowContext->pFlowModel->GetCatchmentCount();

ParamTable \*pLULC\_Table = pFlowContext->pFlowModel->GetTable( "LULC\_A" ); // store this pointer (and check to make sure it's not NULL)

int col\_imperv = pLULC\_Table->GetFieldCol( "IMPERV" ); // get the location of the parameter in the table

int col\_retip = pLULC\_Table->GetFieldCol( "RETIP" ); // get the location of the parameter in the table

int col\_scn = pLULC\_Table->GetFieldCol( "SCN" ); // get the location of the parameter in the table

int col\_scx = pLULC\_Table->GetFieldCol( "SCX" ); // get the location of the parameter in the table

int col\_sc1 = pLULC\_Table->GetFieldCol( "SC1" ); // get the location of the parameter in the table

// iterate through catchments/hrus/hrulayers, calling fluxes as needed

for ( int i=0; i < catchmentCount; i++ )

{

Catchment \*pCatchment = pFlowContext->pFlowModel->GetCatchment(i);

int hruCount = pCatchment->GetHRUCount();

for ( int h=0; h < hruCount; h++ )

{

HRU \*pHRU = pCatchment->GetHRU( h );

int hruLayerCount=pHRU->GetLayerCount();

// pFlowContext->pFlowModel->GetHRUData( pHRU, colLulcB, lulcB, DAM\_FIRST ); // get HRU info

float RETIP,IMPERV,SCN,SCX,SC1=0.0f;

VData lulcA;

pFlowContext->pFlowModel->GetHRUData( pHRU, pLULC\_Table->m\_iduCol, lulcA, DAM\_FIRST ); // get HRU info

if (pLULC\_Table->m\_type==DOT\_FLOAT)

lulcA.ChangeType(TYPE\_FLOAT);

bool ok = pLULC\_Table->Lookup( lulcA, col\_imperv, IMPERV ); // get param value from the table

ok = pLULC\_Table->Lookup( lulcA, col\_retip, RETIP ); // get param value from the table

ok = pLULC\_Table->Lookup( lulcA, col\_scn, SCN ); // get param value from the table

ok = pLULC\_Table->Lookup( lulcA, col\_scx, SCX ); // get param value from the table

ok = pLULC\_Table->Lookup( lulcA, col\_sc1, SC1 ); // get param value from the table

CString msg;

if ( ! ok )

{

msg.Format("Flow plugin couldn't find PRMS parameter RETIP for lulcA %i (looked in column %i)",lulcA, col\_retip);

Report::ErrorMsg(msg);

}

for ( int l=0; l < hruLayerCount; l++ ) //All but the bottom layer

{

HRULayer \*pHRULayer = pHRU->GetLayer( l );

float value = 0;

float d=pHRULayer->m\_volumeWater/pHRU->m\_area;

switch( pHRULayer->m\_layer )

{

case 1:

value = PRMS\_ImperviousRunoff(d, IMPERV, RETIP);

break;

case 2:

value = PRMS\_RechargeZoneRunoff(d,IMPERV, pCatchment->m\_currentThroughFall, SCN, SCX, SC1);

break;

case 4:

value = PRMS\_Interflow(d);

break;

case 5:

value = PRMS\_Baseflow(d);

break;

default:

value=0.0f;

} // end of switch

pHRULayer->m\_contributionToReach=value\*pHRU->m\_area;

pCatchment->m\_contributionToReach+=value\*pHRU->m\_area;

}

}

}

return 0.0f;

}

float WHydro::PRMS\_Interception( float waterDepth, float precip, float &PTF, float COVDN )

{

float STOR = 0.1f;//maximum interception storage depth on vegetation (inches)

STOR=STOR\*2.54f/100.0f;

float XIN = waterDepth;//current depth of interception storage

float PPT = precip;//total precipitation

if (PPT > (STOR-XIN))

PTF = PPT-(STOR-XIN);

float PTN = (PPT \* (1-COVDN))+(PTF\*COVDN); // inches net daily precip on the HRU,

//float EVCAN = 1.0f/100.0f;//evaporation perhaps from pan evaporation data (or from PET calcualtions). This is 1 cm/d

float outflow = PTF;

if (outflow < 0.0f)

outflow=0.0f;

return outflow;

}

float WHydro::PRMS\_Impervious( float waterDepth, float IMPERV, float RETIP )

{

RETIP=RETIP\*2.54f/100.0f;

float RSTOR = waterDepth\*IMPERV;//current depth of impervious storage

float SRO = 0.0f;

float out=0.0f;

if (RSTOR > RETIP)//there is more than the maximum storage

{

SRO=RSTOR-RETIP; //the excess amount will runoff laterally

out = (waterDepth-SRO)\*0.90f;//and the rest will be available for vertical runoff

}

if (out < 0.0f)

out=0.0f;

return out ;

}

float WHydro::PRMS\_RechargeZone( float waterDepth, float IMPERV, float SCN, float SCX, float SC1 )

{

SCN=SCN\*2.54f/100.0f;

float RECHR = waterDepth;

float REMX = 1.0f; //maximum value of RECHR, here in inches;

REMX = REMX\*2.54f/100.0f;

float SRO=0.0f;

//if the volume in the recharge zone is greater than the maximum volume, surface runoff occurs

//we account for that here, but the value for lateral flow is set elsewhere

if (RECHR <= REMX)

{

float CAP = SCN+((SCX-SCN)\*(RECHR/REMX));

SRO = CAP\*RECHR\*(1.0f-IMPERV)\*0.1f;

}

return (RECHR-SRO)\*0.75f;//This is a divergence from PRMS - PRMS does not calculate vertical flow from Recharge Zone

}

float WHydro::PRMS\_LowerZone( float waterDepth, float SMAX )

{

SMAX=SMAX\*2.54f/100.0f;

float storage = waterDepth;

float SEP = 0.50f; //minimum amount of storage not available for recharge

SEP=SEP\*2.54f/100.0f;

float outflow=0.0f;

if (storage > SEP)

outflow = (storage-SEP)\*0.075f;

if (outflow < 0.0f)

outflow=0.0f;

return outflow;

}

float WHydro::PRMS\_SubsurfaceReservoir(float waterDepth, float RCP, float RCF, float RSEP, float RESMX, float REXP )

{

float RES = waterDepth;

float outflow = RSEP\*pow(RES/RESMX,REXP);

return outflow ;

}

float WHydro::PRMS\_GroundwaterReservoir( float waterDepth, float RCB )

{

float GW = waterDepth;

float GSNK = 0.001f;//seepage constant

float SNK = GSNK \* GW;//untracked loss of groundwater

float outflow = SNK;

return outflow;

}

float WHydro::PRMS\_ImperviousRunoff( float waterDepth, float IMPERV, float RETIP )

{

RETIP=RETIP\*2.54f/100.0f;

float RSTOR = waterDepth\*IMPERV;//current depth of impervious storage

float SRO = 0.0f;

if (RSTOR > RETIP)

SRO=RSTOR-RETIP; //the excess amount will runoff laterally

float outflow = SRO;

if (outflow < 0.0f)

outflow=0.0f;

return outflow;

}

float WHydro::PRMS\_RechargeZoneRunoff( float waterDepth, float IMPERV, float PTN, float SCN, float SCX, float SC1)

{

SCN=SCN\*2.54f/100.0f;

float RECHR = waterDepth;

float REMX = 1.0f; //maximum value of RECHR, here in inches;

REMX = REMX\*2.54f/100.0f;

float SRO=0.0f;

//if the volume in the recharge zone is greater than the maximum volume, surface runoff occurs

if (RECHR <= REMX)

{

float CAP = SCN+((SCX-SCN)\*(RECHR/REMX));

SRO = CAP\*RECHR\*(1.0f-IMPERV)\*0.75f;

}

float outflow = SRO;

if (outflow < 0.0f)

outflow=0.0f;

return outflow;

}

float WHydro::PRMS\_Interflow( float waterDepth )

{

float RES = waterDepth;

float RCP = 0.1f;//nonlinear routing coefficient for each subsurface reservoir See Laenen et al 2007

float RCF = 0.001f;//linear routing coefficient for each subsurface reservoir

float RSEP = 0.02f;//daily recharge coefficient

float RESMX = 1.0f;//coefficient

float REXP = 1.5f;//Coefficient

float outflow = (RCF \* RES) + (RCP\*RES\*RES);

return outflow ;

}

float WHydro::PRMS\_Baseflow( float waterDepth )

{

float GW = waterDepth;

float RCB = 0.02f; //gw reservoir routing coefficient

float BAS = RCB \* GW;//baseflow (into stream)

float outflow= BAS;

return outflow;

}

float WHydro::HRUFluxHandler( FlowContext \*pFlowContext )

{

HRULayer \*pHRULayer = pFlowContext->pHRULayer;

Catchment \*pCatchment = pFlowContext->pHRULayer->m\_pHRU->m\_pCatchment;

HRU \*pHRU = pHRULayer->m\_pHRU;

int layer = pHRULayer->m\_layer; // 0 based

float derivative = 0;

// do stuff

float depth = 1.0f; float porosity = 0.4f;

float voidVolume = depth\*porosity\*pHRU->m\_area;

float wc = pHRULayer->m\_volumeWater/voidVolume;

float waterDepth = wc\*depth;

derivative-=GetKTheta(wc);//he bottom soil layer

{

if (layer==pHRU->GetLayerCount()-1)//T

pHRULayer->m\_wc = wc;

float baseflow = GetBaseflow(waterDepth);

derivative-=baseflow;

pCatchment->m\_contributionToReach+=baseflow\*pHRU->m\_area;

}

return derivative\*pHRU->m\_area;

//return 0.0f;

}

float WHydro::GetKTheta(float wc)

{

// For Loamy Sand from HoH.

float wp = 0.035f; float porosity = 0.437f;float lamda = 0.553f ; float kSat = 1.0f;//m/d

float ratio = (wc - wp)/(porosity-wp);

float power = 3.0f+(2.0f/lamda);

float kTheta = (kSat) \* pow(ratio,power);

if (wc < wp) //no flux if too dry

kTheta=0.0f;

return kTheta ;//m/d

}

float WHydro::GetBaseflow(float wc)

{

float k1 = 0.01f;

float baseflow = wc\*k1;

return baseflow;

}

void WHydro::SetGeometry( Reach \*pReach, float discharge )

{

pReach->m\_depth = GetDepthFromQ(pReach, discharge, pReach->m\_wdRatio );

pReach->m\_width = pReach->m\_wdRatio \* pReach->m\_depth;

}

float WHydro::GetDepthFromQ( Reach \*pReach, float Q, float wdRatio ) // ASSUMES A SPECIFIC CHANNEL GEOMETRY

{

// from kellie: d = ( 5/9 Q n s^2 ) ^ 3/8 -- assumes width = 3\*depth, rectangular channel

float wdterm = (float) pow( (wdRatio/( 2 + wdRatio )), 2.0f/3.0f)\*wdRatio;

float depth = (float) pow(((Q\*pReach->m\_n)/((float)sqrt(pReach->m\_slope)\*wdterm)), 3.0f/8.0f);

return (float) depth;

}

float WHydro::ReachFluxHandler( FlowContext \*pFlowContext )

{

Reach \*pReach = pFlowContext->pReach;

ASSERT( pReach != NULL );

ReachSubnode \*pNode = (ReachSubnode\*) pReach->m\_subnodeArray[ pFlowContext->reachSubnodeIndex ];

int hruLayerCount = pFlowContext->pFlowModel->GetHruLayerCount();

float lateralInflow = GetLateralInflow(pFlowContext, pReach->GetSubnodeCount());

float outflow = EstimateReachOutflow(pReach, pFlowContext->reachSubnodeIndex , pFlowContext->timeStep, lateralInflow);

float upstreamInflow = GetInputFlow( pReach, pFlowContext->reachSubnodeIndex );

float derivative = upstreamInflow + lateralInflow - outflow;

pNode->m\_discharge = outflow/86400.0f; //convert units to m3/s

return derivative;

}

float WHydro::GetLateralInflow(FlowContext \*pFlowContext, int subNodeCount)

{

float inflow = 0;

Catchment \*pCatchment = pFlowContext->pReach->m\_pCatchment;

if ( pCatchment != NULL )

inflow = pCatchment->m\_contributionToReach/subNodeCount;//m3

return inflow;

}

// WHydro::EstimateOutflow() ----------------------------------------------

//

// solves the KW equations for the reach downstream from pNode

//------------------------------------------------------------------------

float WHydro::EstimateReachOutflow( Reach \*pReach, int i, float timeStep, float lateralInflow)

{

ReachSubnode \*pNode = (ReachSubnode\*) pReach->m\_subnodeArray[ i ];

float beta = 3.0f/5.0f;

// compute Qin for this reach

float dx = pReach->m\_deltaX;

double dt = timeStep \* 86400.0f;

float lateral = lateralInflow/timeStep;//m3/d

float qsurface = lateral/dx\*timeStep; // units on differentials are m2

float Qnew = 0.0f;

float Qin = 0.0f;

float Q = pNode->m\_discharge;

SetGeometry( pReach, Q );

float width = pReach->m\_width;

float depth = pReach->m\_depth;

//pHydro->waterVolumeArrayPrevious[i]=width\*depth\*length;

float n = pReach->m\_n;

float slope = pReach->m\_slope;

if (slope < 0.001f)

slope = 0.05f;

float wp = width + depth + depth;

float alph = n \* (float)pow( (long double) wp, (long double) (2/3.)) / (float)sqrt( slope );

float alpha = (float) pow((long double) alph, (long double) 0.6);

// Qin is the value upstream at the current time

// Q is the value at the current location at the previous time

Qin = GetInputFlow( pReach, i );

float Qstar = ( Q + Qin ) / 2.0f; // from Chow, eqn. 9.6.4

float z = alpha \* beta \* (float) pow( Qstar, beta-1.0f );

//// start computing new Q value ///

// next, inflow term

float Qin\_dx = Qin / dx \* (float)dt;

// next, current flow rate

float Qcurrent\_z\_dt = Q \* z ;

// last, divisor

float divisor = z + (float)dt/dx;

// compute new Q

Qnew = (qsurface + Qin\_dx + Qcurrent\_z\_dt )/divisor; //m3/s

// pNode->m\_discharge= Qnew;

return Qnew\*86400.0f;

}

float WHydro::GetInputFlow( Reach \*pReach, int subNode )

{

float Q = 0;

if ( subNode == 0)//input flow comes from upstream reaches

{

if ( pReach->m\_pLeft != NULL )

{

int i = pReach->m\_pLeft->GetSubnodeCount()-1;

ReachSubnode \*pNode = (ReachSubnode\*) pReach->m\_pLeft->m\_subnodeArray[ i ];

Q = pNode->m\_discharge;

}

if ( pReach->m\_pRight != NULL )

{

int i = pReach->m\_pRight->GetSubnodeCount()-1;

ReachSubnode \*pNode = (ReachSubnode\*) pReach->m\_pRight->m\_subnodeArray[ i ];

Q += pNode->m\_discharge;

}

}

else // input flows come from upstream subnodes

{

ReachSubnode \*pNode = (ReachSubnode\*) pReach->m\_subnodeArray[ subNode-1 ];

Q = pNode->m\_discharge;

}

return Q;

}

float WHydro::PrecipFluxHandler( FlowContext \*pFlowContext )

{

float time = pFlowContext->time;

HRULayer \*pHRULayer = pFlowContext->pHRULayer;

HRU \*pHRU = pHRULayer->m\_pHRU;

Catchment \*pCatchment = pFlowContext->pHRULayer->m\_pHRU->m\_pCatchment;

// pPrecip->Get(1,time,precip);

//units of precip in m/day to be consistent with Flow

//1 inch/day = 0.0254 m/day

float precip=0.00001f;

if (time > 50.0f && time < 52.0f)

precip = 0.021f;

if (time >=100.0f && time < 104.0f)

precip = 0.0210f;

if (time >=200.0f && time < 201.0f)

precip = 0.0210f\*4.0f;

if (time > 350.0f && time < 355.0f)

precip = 0.021f\*4.0f;

if (time >=425.0f && time < 426.0f)

precip = 0.021f\*4.0f;

if (time >=520.0f && time < 540.0f)

precip = 0.0254f\*4.0f ;

pCatchment->m\_currentPrecip=precip;

return precip\*pHRU->m\_area;// m3/d

}

float WHydro::ETFluxHandler( FlowContext \*pFlowContext )

{

float time = pFlowContext->time;

HRULayer \*pHRULayer = pFlowContext->pHRULayer;

HRU \*pHRU = pHRULayer->m\_pHRU;

int layer = pHRULayer->m\_layer; // 0 based

//units of precip in m/day to be consistent with Flow

//1 inch/day = 0.0254 m/day

float area = pHRU->m\_area;

float ET=0.00254f;

if (pHRULayer->m\_wc\*2.0f < ET)

ET = pHRULayer->m\_wc\*0.005f;

if (pHRULayer->m\_wc<=0.01f)

ET = 0.0f;

if (layer == 1)

area=area\*0.05f;//for the impervious zone we make an assumption about the percent of the HRU that is impervious

if (layer == 3)

ET = ET\*0.5f;//only transpiration is taken from the lower zone of the soil zone reservoir

return ET\*area;// m3/d

}

float WHydro::NetworkHandler( FlowContext \*pFlowContext )

{

FlowModel \*pModel = pFlowContext->pFlowModel;

int reachCount = (int) pModel->m\_reachArray.GetSize();

for ( int i=0; i < reachCount; i++ )

{

Reach \*pReach = pModel->m\_reachArray[ i ];

pFlowContext->pReach = pReach;

for ( int l=0; l < pReach->GetSubnodeCount(); l++ )

{

Reach \*pReach = pFlowContext->pReach;

ASSERT( pReach != NULL );

ReachSubnode \*pNode = (ReachSubnode\*) pReach->m\_subnodeArray[ l ];

float lateralInflow = GetLateralInflow(pFlowContext, pReach->GetSubnodeCount());

float outflow = EstimateReachOutflow(pReach, l , pFlowContext->timeStep, lateralInflow);

pNode->m\_discharge = outflow/86400.0f; //convert units to m3/s

}

}

return -1.0f;

}

float WHydro::Init(FlowContext \*pFlowContext)

{

FDataObj \*m\_pClimateData = new FDataObj;

m\_pClimateData->ReadAscii("C:\\Envision\\StudyAreas\\WESP\\climateCorvallisDaily.csv" , ',', TRUE );

return -1.0f;

}