

Water Quality Engine Technical Manual

Water Quality Engine Technical Manual

Exported on
10/18/2022

Table of Contents

1	Search this documentation	5
2	Popular Topics	6
3	Featured Pages.....	7
4	Recently Updated Pages.....	8
5	Overview	9
5.1	Objectives.....	9
5.2	Software Summary	10
5.3	Compilation Information and Dependencies	11
6	Overview of a Typical Water Quality Simulation	12
6.1	Initialization	12
6.1.1	Load Files.....	12
6.1.2	Geometry Creation.....	12
6.1.3	Runtime Window.....	14
6.1.4	Water Quality Constituent Data	15
6.1.5	Initial Conditions.....	15
6.1.6	Boundary Conditions.....	15
6.1.7	Meteorological Data.....	15
6.1.8	Initial Cell Volumes	15
6.2	Program Flow Control and Data Exchange.....	16
6.3	Finalization	16
7	Java Implementation.....	17
7.1	ResSim Water Quality Test Driver.....	17
7.2	Java Water Quality Library Classes	17
8	Generalized Water Quality Engine Description	19
8.1	Stability Limitations and Substepping	20
8.2	River Hydraulics	21
8.2.1	Dispersion.....	22
8.3	Reservoir Hydraulics	23
8.3.1	Equation of State	25

8.3.2 Inflow Allocation	26
8.3.3 Withdraw Allocation	26
8.3.4 Effective Vertical Diffusion.....	30
8.3.5 Mixing.....	31
9 Initial Verification Applications	32
9.1 Transport of a Conservative Constituent through ResSim Network.....	32
9.2 Reservoir Water Temperature Simulation	36
10 References	41
11 Appendix A: Java API For WQEngineAdapter	42
11.1 Enumerated Types.....	42
11.2 Native Methods	42
11.3 Java Methods	48
12 Appendix B: Fortran API	53

DRAFT. FOR LIMITED USE ONLY. DO NOT DISTRIBUTE.DRAFT. FOR LIMITED USE ONLY. DO NOT DISTRIBUTE.



DRAFT Technical Manual for the Generalized One-Dimensional Reservoir and River Water Quality Engine Prototype. DO NOT DISTRIBUTE. FOR LIMITED USE ONLY.

1 Search this documentation

2 Popular Topics

No labels match these criteria.






3 Featured Pages

Content by label

There is no content with the specified labels



4 Recently Updated Pages

-  [Water Quality Engine Technical Manual](#)(see page 4)
Mar 01, 2022 • updated by Sara O'Connell¹ • view change²
-  [Generalized Water Quality Engine Description](#)(see page 19)
Feb 01, 2022 • updated by Sara O'Connell³ • view change⁴
-  [Generalized Water Quality Engine Description](#)(see page 19)
Jan 13, 2022 • updated by Alex Davis⁵ • view change⁶
-  [Overview of a Typical Water Quality Simulation](#)(see page 12)
Jan 11, 2022 • updated by Alex Davis⁷ • view change⁸
-  [Appendix B: Fortran API](#)(see page 53)
Dec 20, 2021 • updated by Sara O'Connell⁹ • view change¹⁰

¹ <https://www.hec.usace.army.mil/confluence/display/~soconnell>

² <https://www.hec.usace.army.mil/confluence/pages/diffpagesbyversion.action?pagelId=78875206&selectedPageVersions=6&selectedPageVersions=7>

³ <https://www.hec.usace.army.mil/confluence/display/~soconnell>

⁴ <https://www.hec.usace.army.mil/confluence/pages/diffpagesbyversion.action?pagelId=78873182&selectedPageVersions=50&selectedPageVersions=51>

⁵ <https://www.hec.usace.army.mil/confluence/display/~adavis>

⁶ <https://www.hec.usace.army.mil/confluence/pages/diffpagesbyversion.action?pagelId=78873182&selectedPageVersions=48&selectedPageVersions=49>

⁷ <https://www.hec.usace.army.mil/confluence/display/~adavis>

⁸ <https://www.hec.usace.army.mil/confluence/pages/diffpagesbyversion.action?pagelId=78873164&selectedPageVersions=22&selectedPageVersions=23>

⁹ <https://www.hec.usace.army.mil/confluence/display/~soconnell>

¹⁰ <https://www.hec.usace.army.mil/confluence/pages/diffpagesbyversion.action?pagelId=78873215&selectedPageVersions=6&selectedPageVersions=7>

5 Overview

5.1 Objectives

The generalized water quality engine was developed to fulfill two objectives.

1. To begin the process of adding water quality modeling capabilities to the major US Army Corps of Engineers Hydrologic Engineering Center (HEC) watershed modeling software products, including HEC-RAS (River Analysis System), HEC-ResSim (Reservoir System Simulation), and HEC-HMS (Hydrologic Modeling System). The water quality engine is an independent library of routines, written in Fortran, which is called to calculate the transport and transformation of water quality constituents by the individual driving programs. This allows for a consistent treatment of water quality transport, input and output, and interfacing with the pre-existing water quality libraries developed by the USACE Engineering Research and Development Center Environmental Lab (ERDC-EL). This framework will facilitate the linking of different models to simulate a range of conditions from upper watershed basins, to reservoir systems, to lower river systems.
2. To streamline the process of modeling of reservoir releases for water quality objectives. The initial application of the generalized water quality engine will be to the HEC-ResSim software. Current methods of modeling reservoir releases for environmental objectives are labor intensive and include simulating flows using release decision software such as ResSim, then using those flows to drive a water quality simulation, often using different software, and then using the water quality model results to adjust and re-run the reservoir release simulation. By integrating water quality modeling capabilities directly into the ResSim software, the iterative process of determining reservoir releases for water quality objectives can be directly addressed.

This document describes the first phase (prototype) implementation of the generalized water quality engine. Rivers are represented using one-dimensional (1D) horizontal segments, which are assumed to be laterally and vertically well-mixed. Reservoirs are represented using 1D segments, which are assumed to be laterally and longitudinally well-mixed. The driving program (ResSim, RAS, or HMS) provides initial setup information, including:

- geometry (river-reservoir network connectivity, discretization information)
- water quality constituents to be simulated (e.g., water temperature, dissolved oxygen, and/or nutrients)
- initial conditions (distributions of all constituents at the start of the simulation)
- water quality constituent parameters (coefficients used in the calculation of constituent transformations)
- dispersion coefficients (parameters describing longitudinal or vertical mixing rates)

and forcing information, including:

- compute flow control
- hydrodynamic information (e.g., flows, conveyance areas, reservoir storage)
- water quality boundary concentrations for inflows
- meteorological data

and the water quality engine calculates the transport and transformations of the water quality constituents. Transport is calculated by numerically solving the advection-diffusion-reaction equation on a discretized version of the watershed geometry. Transformations are calculated using ERCD water quality libraries.

For the prototype application, a test driver application is used as a proxy for the full ResSim program, and only a conservative scalar constituent and water temperature are available for modeling. Later phases of development of the generalized water quality engine prototype will extend modeling capabilities to additional water quality constituents, (e.g., dissolved oxygen, nutrients, phytoplankton) driving software (e.g., HEC-HMS, HEC-RAS), and higher dimension systems (e.g., 2D longitudinally-segmented reservoirs and 2D RAS regions).

This work has been performed under USACE contract W91238-14-D-0005, order number 0068 under the direction of Dr. Todd Steissberg, Senior Hydraulic Engineer at the USACE Hydrologic Engineering Center.

5.2 Software Summary

The water quality engine is written using the Fortran programming language. This language was chosen for two reasons: 1) the computational speed of the language for problems involving large watersheds, highly resolved areas, or large numbers of water quality constituents, and 2) interfacing with pre-existing ERDC-EL water quality libraries, which are also written in Fortran.

Communication between the driving program (ResSim, written in Java) and the water quality engine utilizes the Java Native Interface (JNI), a framework which allows Java code to call libraries written in platform-dependent languages such as C or C+. For the water quality engine, Fortran API routines, bundled as a static library (.lib file) are called through C++ wrappers, which are bundled as a dynamic library (.dll). The JNI and C++ wrappers pass arrays of primitive variables (e.g., integer and double types) for efficient communication and to prevent unnecessary array copying. From the Java side, a specific class (WQEngineAdapter) provides Java calls to access the native methods.

The WQEngineAdapter class is part of a package of Java classes that provide support for interaction with the water quality engine. It is intended that these routines will be utilized by other driving programs written in Java in order to run the water quality engine. Also included in this package are classes which:

- discretize an element-node style ResSim geometry containing stream reaches and reservoirs into a series of water quality cells and cell boundaries connecting adjacent cells (referred to in this document as "faces")
- set information about water quality constituents, specific parameters for modeling those constituents, and initial conditions
- set forcing variables for water quality transport and transformations (e.g., flows, boundary conditions, meteorological information)
- control compute flow, including setting and reloading the state of the system (a snapshot of all water quality constituent concentrations at a particular time) for use with iterative computes
- retrieve results from the water quality engine and output them to a HDF5 file

Relationships between the software components are shown in the figure below (see page 10).

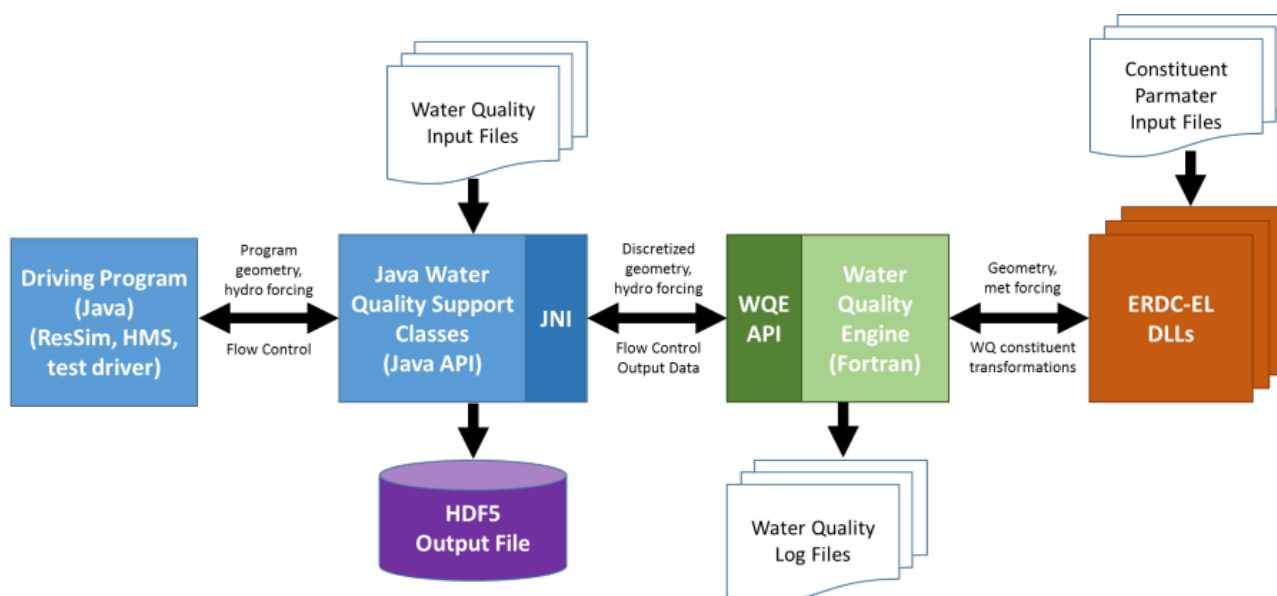


Figure: Interaction between water quality driving program, Java support classes, generalized water quality engine, and ERDC-EL water quality libraries.

5.3 Compilation Information and Dependencies

The Fortran .lib file is compiled using the Microsoft Visual Studio 2015, Update 3 integrated development environment with Intel Parallel Studio 2017, Update 2. The C++ wrapper library is compiled using Microsoft Visual Studio 2017 with Visual C++ 2017 edition. The Java water quality support classes utilize Java 8 features and depend on hecjavadev v5.0 (geometry, data, and heclib.util packages). Input and output to HDF5 files in Java requires dependence on classes in the H5_Interface_Java library, written by Todd Steissberg, HEC. The water quality engine .dll, any ERDC-EL .dll's utilized during the simulation, and several HDF5 IO .dll's must be in specified directories available at runtime.

6 Overview of a Typical Water Quality Simulation

The following sections describe the flow of a typical water quality simulation. Specific Java API calls are given in bold. Appendix A details the Java API and gives specific information about input and output parameters for each call.

6.1 Initialization

The water quality compute begins with the initialization of the required .dll's, opening of output files, creation and input of water quality geometry, and input of information about the water quality constituents to be modeled and their initial concentration distributions.

6.1.1 Load Files

Initialization starts by loading the water quality engine .dll, opening its log file, and setting the unit system (English or metric) for the compute (**initCompute**). If reach elements are included in the simulation, a RAS steady-flow output table file must be loaded. This table file gives hydrodynamic properties (e.g., conveyance area and surface width) as a function of flow magnitude at specific cross-sections, and is created from the results of a series of steady flow RAS model simulations (for more information, see later section on River Hydraulics).

6.1.2 Geometry Creation

The Java water quality engine support classes take an element-node style ResSim geometry and convert it into a group of water quality subdomains, each discretized into a series of water quality cells. Boundaries between adjacent cells or at cell inflow or outflow boundaries are referred to as cell faces (or simply “faces”). The driving program must provide a sorted list of reach and reservoir elements in the network, where upstream elements are provided first in the list and downstream elements are given last. An example ResSim network composed of reach elements (blue lines), reservoir elements (blue shapes), and stream junctions (red circles) is given in [figure below](#) (see [page 13](#)). Local flow and diversions are denoted by black circles surrounding junction elements. From this example geometry, five subdomain objects are created, each defined by their boundary junctions. Seven water quality subdomain boundary objects are created which link the water quality subdomains to other water quality subdomains or global boundaries. In this example, the global boundaries include two upstream inflows and one outflow. Two boundary objects are defined at the stream confluence above the reservoir, with one connection the downstream reach for each of the upstream reaches.

The breakup of the water quality modeling domain into subdomains allows for more efficient computations. Upstream subdomains may be computed independently for the entirety of the simulation if they are unaffected by reservoir release decisions. Subsequent simulation iterations to determine reservoir releases from water quality control rules can then take place on a subset of the model domain, saving computational time. If flows or other forcing functions change due to reservoir operations, only those subdomains downstream of the change will be affected and need to be recomputed.

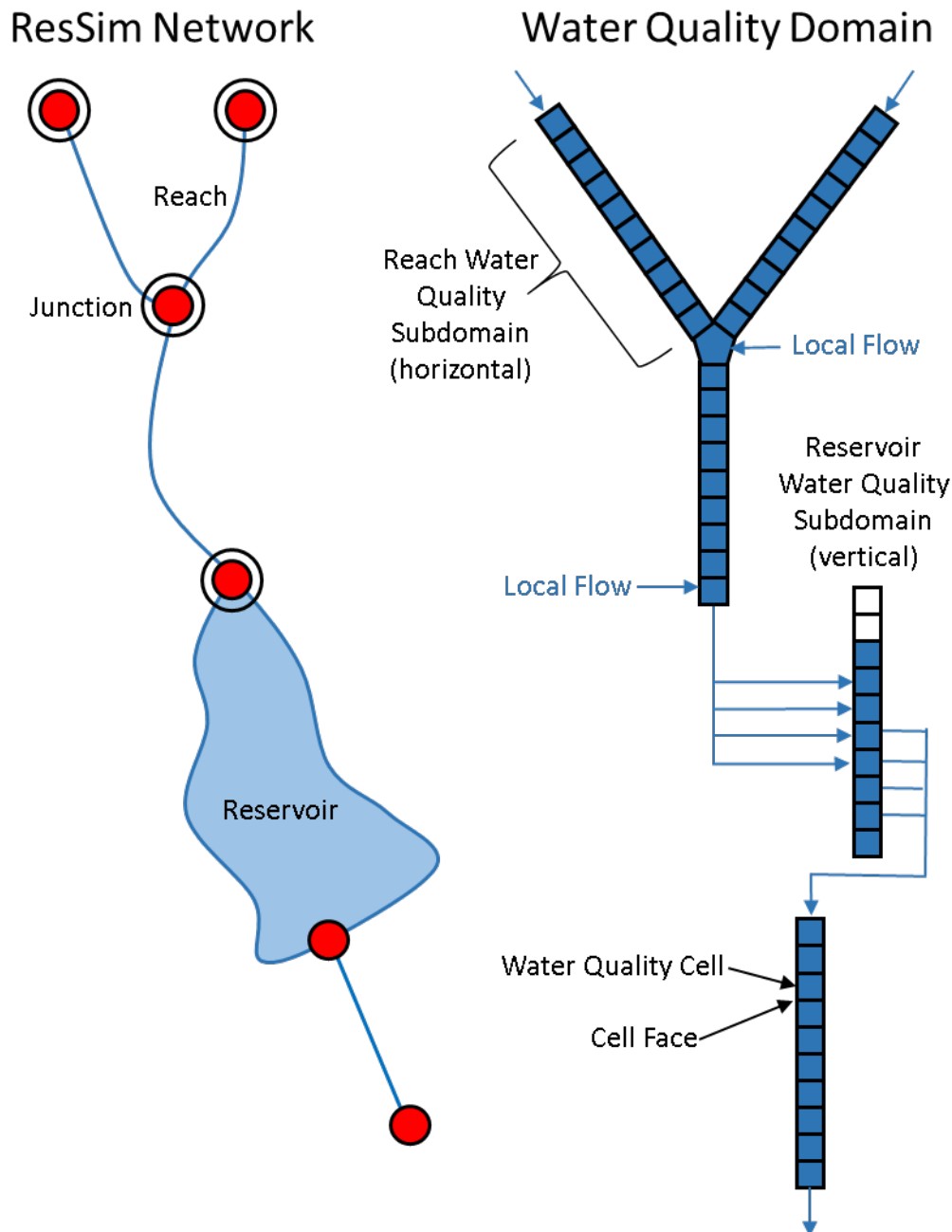


Figure: Example ResSim network composed of stream reaches and a reservoir, and its corresponding discretized water quality domain.

Each of the reach subdomains is discretized into a user-determined number of water quality cells. Each water quality cell initially has two faces, connecting its upstream and downstream ends to adjacent water quality cells. The water quality subdomain has n_{cell} total water quality cells, and $n_{\text{cell}}-1$ internal faces. The most upstream face is linked to the upstream subdomain (or specified as a global inflow boundary), and a similar process links the most downstream face to the downstream subdomain. Channel confluences/bifurcations are handled by adding boundary faces to the most upstream/downstream cells.

Local inflows at junctions are treated by adding boundary faces to the water quality cells immediately downstream of the junction when the junction joins two reaches, or immediately upstream of the junction when the junction joins an upstream reach and downstream reservoir (see [figure above](#)(see [page 13](#))). Additional boundary objects are

created to hold information about these local flows and diversions. By appending boundary faces at end of the cell face array, the compute logic can quickly loop through the internal faces without having to check if they are boundary faces. This avoids the use of branching logic in the Fortran code, which can slow down compute times.

Reservoir subdomains are similarly discretized into a user-determined number of water quality cells. In this case, the water quality cells represent the vertical layers. Reservoirs are discretized from their minimum to maximum elevation. Upper layers are allowed to become dry when the reservoir is not completely full. No additional boundary faces are added to reservoir cells, as inflows and outflows may come from any of the layers. Discretization information includes:

- Number of cells
- Number of faces
- Number of boundary faces
- Bounding face indexes for each cell
- Bordering cell indexes for each face
- Subdomain type:
 - reservoir
 - reach
- Reservoir elevation-area and elevation-storage curve data
- Cell center (x, y, z) locations
- Boundary indexes for boundary faces
- Cell length (reach) or height (reservoir) for each cell

These values are passed to the water quality engine in **setGeoData** for every subdomain.

Boundary condition data includes:

- Upstream and downstream subdomain indexes that the boundary connects
- Boundary type:
 - Reach inflow boundary
 - Reservoir inflow boundary
 - Reach outflow boundary
 - Reservoir outflow boundary
 - River reach to river reach junction
 - River reach to reservoir junction
 - Reservoir to river reach junction
 - Reservoir to reservoir junction
 - Local inflow boundary
 - Local diversion boundary

The boundary condition data is then passed to water quality engine in **setBoundaryGeoData** for every boundary.

6.1.3 Runtime Window

The run time window information consists of the simulation start time, the simulation end time, the driving program simulation time step, and is set in **setRunTimeInfo**. In ResSim, the simulation begins with a lookback period to initialize flows in order to create a realistic starting condition for use in the forecast simulation. However, as reservoir volume is not conserved during this period (the sum of the reservoir inflows and outflows are not equal to change in storage), no water quality computations are performed for the lookback period, and the water quality simulation start time is defined as the start of the forecast period. The simulation time step defines the interval at which hydrodynamic information is passed to the water quality engine. However, if the water quality cell size, flows, or constituent reaction rates require, sub-time steps at intervals less than the simulation time step are used for the transport of water quality constituents (see later section Stability Limitations and Substepping).

6.1.4 Water Quality Constituent Data

The user-defined input for the number and types of water quality constituent to be simulated are passed to the engine in **setWQConstituentData**. For this prototype, only conservative scalar and water temperature constituents are supported. If the temperature is being modeled, the water quality engine initializes an ERDC-EL temperature .dll for the calculation of boundary heat fluxes. At this point, an input files which lists temperature model parameter values (e.g., coefficients for calculating evaporative cooling as a function of wind speed) is read in by the ERDC-EL .dll.

6.1.5 Initial Conditions

User-defined initial conditions are set using **setParam**. Initial conditions are assumed to be zero for all water constituents and subdomains unless otherwise specified.

6.1.6 Boundary Conditions

The time series of concentrations for each water quality constituent for each inflow boundary are passed into the water quality engine in **setBoundaryConcentrations**. If no data is prescribed, the inflow arrays are assumed to be zero. In the current prototype implementation, data for all time steps in the simulation are passed to the engine at the start of the simulation.

6.1.7 Meteorological Data

If water temperature is simulated, the user must provide data records for the following meteorological variables:

- Solar radiation (W m^{-2})
- Air temperature ($^{\circ}\text{C}$)
- Wind speed (m s^{-1})
- Cloud cover (fraction in range 0–1)
- Atmosphere pressures (atmospheres)
- Air water vapor pressure (millibars)

Each of the meteorological variables are set for a user-defined number of meteorological stations and, similar to the treatment of BCs, meteorological data for all time steps is transferred to the water quality engine at the start of the simulation. This is done using **setMetData**.

6.1.8 Initial Cell Volumes

ResSim calculates flows through river reaches, but does not calculate water volumes in reaches. Because of conservation of mass, the water quality engine needs to store and update volumes in the water quality cells. This is done automatically for every time step in the simulation after the first one, using the inputs of flow at the water quality cell faces. However, the water volume in the cells for the initial time step needs to be set by the user (**setInitialCellVolumes**). It is not necessary to do this for reservoir subdomains, as storage volume is passed to the engine in at every time step. For the prototype application, reach subdomain cell volumes are computed using the average conveyance area at the faces for the first time step, multiplied by the length of the cell.

6.2 Program Flow Control and Data Exchange

The Java interface call **compute** controls the water quality simulation stepping. Prior to this, hydro forcing variables must be updated for the start and end time of the timestep as defined in **updateHydro**. These include:

- Reaches
 - Flow at faces
 - Surface width at faces
 - Conveyance area at faces
- Reservoir
 - Storage
 - Seepage flow
 - Net evaporation and precipitation flow
 - Inflows and outflows

In ResSim, the flow at the subreaches within a reach is computed depending on the stream routing method chosen. These flows are then interpolated to the water quality cell faces. The input from the RAS steady flow simulations is used as a lookup table to get conveyance area and face surface width at a given flow for a given face. After the hydrology variables are set, the water quality transport computation can take place for a single time step. In ResSim, it is often necessary to compute several steps forward in time, then use those results to update reservoir release decisions, and then re-compute the time window. To facilitate this non-linear, looping compute flow, the API routines **saveState** and **restoreState** were created to save the given state of the system (the concentration of all water quality constituents for all water quality cells) and then reload it when looping backward in time. When performing a sequential compute (e.g., an "offline" run uncoupled to the hydrodynamics), setting the hydrodynamic variables only needs to be done once per compute step. For these simulations, internal Fortran pointers for the end-of-step hydrodynamic arrays simply have their target changed to indicate start-of-step values. In this way unnecessary data passing and copying is prevented.

The API routine **getParam** is used to get a water quality constituent parameter (e.g., water temperature, scalar concentration, or scalar mass) at a given location and time for use with reservoir release decision making. **saveSnapshot** outputs user-defined parameters for each cell to a HDF5 output file, utilizing the WQIO class. These parameters include:

- Hydrodynamic information
 - Cell volumes
 - Face velocities
 - Face flows
- Water quality constituents
 - Scalar concentrations
 - Cell scalar masses
 - Water temperature (°C)

6.3 Finalization

The Java API method **finalizeCompute** unloads the DLLs, deallocates arrays, and closes the log and output HDFS files.

7 Java Implementation

7.1 ResSim Water Quality Test Driver

The water quality test driver utilizes the run-from-script ("headless") functionality of ResSim to simulate any number of alternatives and watersheds listed in an .xml input file. For each alternative, the script runs the ResSim forecast compute first, then water quality simulation is computed over the same time period. For future testing and verification of the water quality engine with new development, the script can keep track of failed tests and write simple output reports.

7.2 Java Water Quality Library Classes

A short description of the Java water quality library classes is given below. The [figure below](#)(see page 18) shows a UML class diagram of the package.

- **WQTest** – Runs a specific water quality simulation.
- **WQEngineAdapter** – Provides a link to the Fortran water quality libraries.
- **WQIO** – Manages output of water quality geometry and results to a HDF5 file.
- **WQGeometry (RssWQGeometry)** – Holds a list of water quality subdomain and boundary objects. Contains methods for creating a water quality geometry from a ResSim network and extracting flow information for input to the WQEngine.
 - **WQGeoSubdomain** – Holds information about discretization (face and cell info) and connectivity of a subdomain. Contains methods for discretizing a reach or reservoir subdomain from length or height information.
 - **WQReachHydro** – Holds sub-reach hydrodynamic properties (flow, conveyance area, surface width) for a reach subdomain.
 - **WQResHydro** – Holds hydrodynamic properties (storage, seepage, evaporation, inflows, outflows) for a reservoir subdomain.
 - **WQGeoSubdomainBoundary** – Holds information about boundaries connecting subdomains and external inflows and outflows.
- **WQTime (RssWQTime)** – Holds information about the water quality simulation time window.
- **WQMetStationSet** – Holds list of meteorological stations.
 - **WQMetStation** – Holds meteorological station information.
 - **WQMetDataSet** – Holds meteorological data sets for a given met station.
- **WQConstituentSet** – Holds a set of WQConstituent objects to be modeled.
 - **WQConstituent** – Holds information about a water quality constituent.

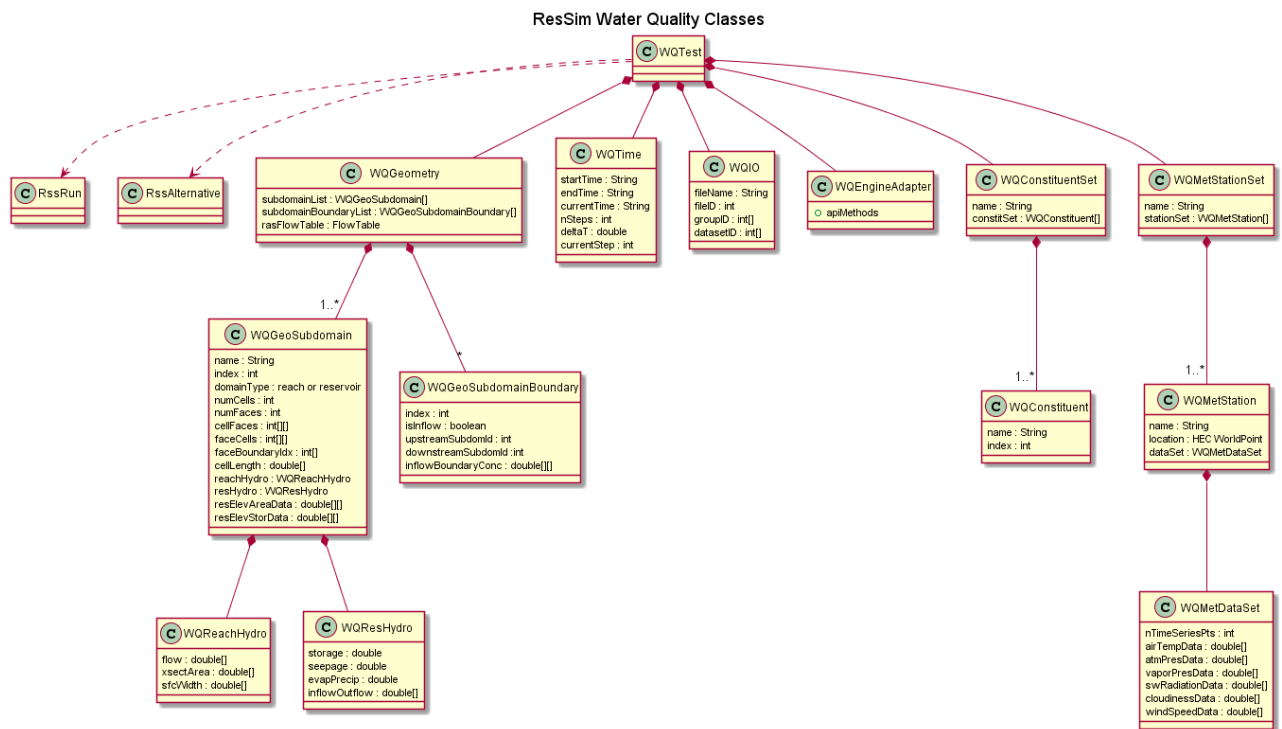


Figure: Class diagram showing Java classes and class variables involved in a test water quality simulation run.

8 Generalized Water Quality Engine Description

The water quality engine computes the numerical solution to the advection-diffusion-reaction equation for the transport and transformation of a water quality constituent. This [equation](#)(see [page 0](#)) is given in 1D form below.

$$\underbrace{\frac{\partial C}{\partial t}}_{\text{time derivative}} + \underbrace{\frac{\partial(vC)}{\partial x}}_{\text{advection}} = \underbrace{\frac{\partial}{\partial x}\left(K \frac{\partial c}{\partial x}\right)}_{\text{diffusion}} + \text{sources/sinks} \quad \text{Equation 1}$$

C is the concentration of a water quality constituent, v is the water velocity in the x direction, and K is a dispersion or diffusion coefficient, parameterizing the amount of mixing between adjacent cells. For reach subdomains, x is the streamwise direction and K represents longitudinal dispersion. For reservoir subdomains, x is the vertical direction and K represents an effective vertical diffusion.

For transport of a conservative scalar, the source-sink term is identically zero. For water temperature, the source-sink term is

$$\text{sources/sinks} = \frac{H_{\text{net}} A}{V \rho C_p} \quad \text{Equation 2}$$

where A is the surface area, V is the water volume, ρ is the water density, C_p is the specific heat of water (4179 J kg⁻¹ °C⁻¹), and H_{net} (Watts m⁻²) is the net heat flux, which includes:

- shortwave solar radiation,
- longwave downwelling and back radiation,
- latent or evaporative heat flux,
- sensible or convective heat flux, and
- bed conduction (optional).

Each subdomain is discretized into a grid composed of water quality cells—1D horizontal cells for reaches and 1D vertical layers for reservoirs—and solved using a finite volume method. Variables are defined using a staggered grid approach, where the hydrodynamic parameters of cell volume (V) and surface area (A) are defined at the cell centers (i, i+1 in the [figure below](#)(see [page 19](#))), and velocity (v), flow (Q), conveyance area (a), and diffusion coefficients are defined at the cell faces (j in Figure 4). Water quality concentration values are defined at the cell centers.

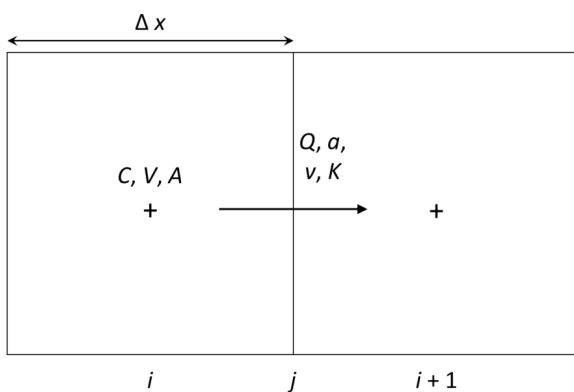


Figure: Variable definition sketch for a staggered water quality grid.

Constituent fluxes are calculated as

$$flux_j = \underbrace{Q_j C_j}_{\text{advection}} + \underbrace{K a \frac{(C_i - C_{i+1})}{\Delta x}}_{\text{dispersion}} \quad \text{Equation 3}$$

In this prototype implementation, the concentrations at the cell faces (C_j) are determined using a first order upwind method

$$C_j = \begin{cases} C_i & \text{if } v_j \geq 0 \\ C_{i+1} & \text{otherwise} \end{cases} \quad \text{Equation 4}$$

Second order flux-limiting methods will be introduced to better handle transport in advective environments in future phases of development. Once the fluxes are computed at the faces, cell concentrations are updated according to conservation of mass

$$C_i^{n+1} V_i^{n+1} = C_i^n V_i^n + \Delta t \sum_{j=1}^{n_{faces}} flux_j \quad \text{Equation 5}$$

where the superscript n denotes values at the beginning of the time step, and $n+1$ denotes values at the end of the time step. Source and sink fluxes are computed at the end of the time step using the updated cell concentrations and volumes.

In order to get the cell volumes at the end of the time step, the continuity equation must be solved

$$V_i^{n+1} = V_i^n + \Delta t \sum_{j=1}^{n_{faces}} Q_j \quad \text{Equation 6}$$

This is done for all cells in the reach subdomains using the start-of-step cell volumes and the flows passed to the engine. For reservoir subdomains, only the top most active layer will need a volume update. It is clear from [Equation 5](#) (see page 20) and [Equation 6](#) (see page 20) that the hydrodynamic information passed to the water quality engine from the driving program must satisfy the continuity equation in order to ensure the conservation of mass of the water quality constituents. More information on how this is accomplished in reach subdomains is given in the section on River Hydraulics.

8.1 Stability Limitations and Substepping

The finite volume method described above is an explicit numerical method and is subject to restrictions on the length of the time step in order to prevent cell overdrafting.

$$\underbrace{\frac{v \Delta t}{\Delta x}}_{\text{Courant number restriction}} + \underbrace{\frac{K \Delta t}{(\Delta x)^2}}_{\text{Diffusion number restriction}} < 1 \quad \text{Equation 7}$$

The first term on the left is known as the Courant or CFL number and quantifies the stability restriction due to advective flux. The second term is the Diffusion number and quantifies the restriction due to diffusive flux. In order to not violate this stability condition, maximum allowable time steps are computed at the start of every computational step based on that step's velocities and dispersion/diffusion coefficients. If the maximum allowable time step is less than the hydrodynamic model time step, the number of substeps is calculated by ceiling $\left(\frac{\Delta t_{hydro}}{\Delta t_{max}} \right)$.

The constituent transport equation ([Equation 1](#) (see page 0)) and conservation of cell volume equation ([Equation 6](#) (see page 0)) are solved, and cell concentrations are updated, for every substep.

8.2 River Hydraulics

A river reach is discretized in the water quality model as shown in the [figure below](#) (see [page 21](#)). Local inflows and diversions at stream junctions are typically placed at the upstream-most water quality cell in the subdomain downstream of the junction. If there are multiple reaches downstream of the junction, the local flow is uniformly distributed into all of the upstream-most cells in the subdomains. When a reach enters a reservoir through a junction containing a local inflow or diversion, the local flow is applied at the downstream-most cell in the reach subdomain rather than treated as an individual inflow entering the reservoir.

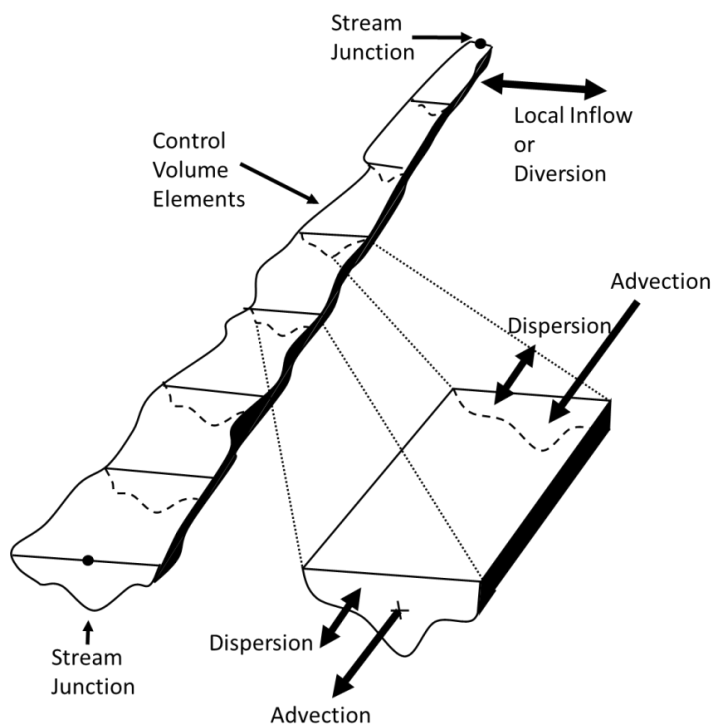


Figure: Discretization of a river reach subdomain.

To solve the constituent transport equation, the water quality engine requires input of flows at all water quality cell faces. These are used to calculate constituent fluxes ([Equation 3](#) (see [page 0](#))) and update cell volumes ([Equation 6](#) (see [page 0](#))). In a ResSim simulation, flows are calculated at the stream junctions, which correspond to only the most upstream and downstream cell faces in the subdomain. Flows may also be calculated at locations within the reach (subreaches) if one of a few options for the reach's hydrologic routing method are specified by the user: Muskingum, Modified-Puls, SSARR, or Working R&D. In these cases, the reach is equally divided by a given number of subreaches, and subreach flows are calculated by ResSim using some variation of the continuity equation. Arrays of subreach flow values are stored for the previous and current time step in ResSim, but are not output to a file. For an offline water quality simulation, the reach routing objects therefore need to be re-initialized and subreach flows need to be recomputed for each time step. Because consistency with the continuity equation is required to maintain conservation of constituent mass in the water quality engine, this prototype application uses the ResSim-calculated subreach flows exactly. The assumption is therefore made that the number of water quality cells in a particular reach subdomain equals the number of subreaches used for routing. In future phases of development, this requirement will be relaxed and any necessary interpolation of ResSim subreach flows to water quality face flows will be done in a manner that maintains conservation of volume.

Another aspect of ResSim's hydrologic routing methods is that only flow rates are calculated; volumes in the reaches or subreaches are never explicitly computed. However, cell volumes are required for the water quality engine ([Equation 5](#)(see page 0)). This discrepancy is handled by the water quality engine by calculating the initial cell volumes by multiplying the longitudinal cell length by the average face conveyance area. Volumes are updated for every subsequent time step using input face flows and [Equation 6](#)(see page 0).

There are additional hydrodynamic properties that are required by the water quality engine for the calculation of heat fluxes and dispersion coefficients (e.g., face conveyance area, cell surface area). The most realistic approach to determining these quantities is to utilize HEC-RAS results. The user must have access to a RAS geometry which covers all of the ResSim reach elements where water quality will be modeled. RAS must then be run in steady flow mode over a series of flow values which span the expected range of flows for the simulation. Once complete, the user must export a Profile Output table in the proper format, for input to the water quality model. An example of this output table is given in the [figure below](#)(see page 23) for a sample RAS geometry with a single river, 343 cross-sections, and 16 steady flow values.

In the current prototype implementation of the water quality engine, a RAS steady flow profile output table is read in, and a lookup table is created in Java to obtain cross-sectional hydraulic properties for any given flow value. However, because of the geometry linking that must occur between the locations of RAS cross-sections and water quality cell faces, something that is more easily accomplished with user-interface (UI) map tools, the use of the RAS hydraulic properties was postponed for implementation in tandem with the ResSim water quality UI development. Current hydraulic properties in the engine are set to constant, non-flow dependent values.

8.2.1 Dispersion

Reach dispersion values (K_h) are calculated at the interior cell faces based on the following equations

$$K_h = 0.011 \frac{v^2 W^2}{dv^*} \quad \text{Equation 8}$$

$$v^* = v \sqrt{gd} \quad \text{Equation 9}$$

where

v^*	= shear velocity
W	= stream top width
g	= gravitational acceleration
d	= hydraulic depth

No dispersion is calculated at the boundary faces. This is the same formulation used in the current RAS 1D water quality model, and is described in Fisher et al. (1979).

Profile Output Table - Standard Table 1

HEC-RAS Plan: Example for ResSim WQ River: Minnesota Reach: Lower

Rivers = 1
 # Hydraulic Reaches = 1
 # River Stations = 343
 # Plans = 1
 # Profiles = 16

Reach	River Sta	Profile	Q Left (m3/s)	Q Channel (m3/s)	Q Right (m3/s)	Flow Area L (m2)	Flow Area Ch (m2)	Flow Area R (m2)	Vel Left (m/s)	Vel Chnl (m/s)	Vel Right (m/s)	Top W Left (m)	Top W Chnl (m)	Top W Right (m)
Lower	58441.4	PF 1		50.00			114.87			0.44			63.99	
Lower	58441.4	PF 2		100.00			165.31			0.60			68.19	
Lower	58441.4	PF 3		150.00			206.54			0.73			71.44	
Lower	58441.4	PF 4		200.00			243.48			0.82			74.15	
Lower	58441.4	PF 5		250.00			277.22			0.90			75.20	
Lower	58441.4	PF 6		300.00			308.75			0.97			76.17	
Lower	58441.4	PF 7		350.00			338.39			1.03			77.00	
Lower	58441.4	PF 8		400.00			366.69			1.09			77.93	
Lower	58441.4	PF 9		450.00	0.00		393.61	0.12		1.14	0.01		78.73	2.47
Lower	58441.4	PF 10		499.93	0.07		419.55	2.55		1.19	0.03		79.49	15.09
Lower	58441.4	PF 11		549.61	0.39		444.41	10.82		1.24	0.04		79.92	41.74
Lower	58441.4	PF 12		598.72	1.28		468.08	27.10		1.28	0.05		80.07	63.36
Lower	58441.4	PF 13		647.23	2.77		490.22	46.47		1.32	0.06		80.20	76.81
Lower	58441.4	PF 14		695.32	4.68		511.20	70.83		1.36	0.07		80.33	100.64
Lower	58441.4	PF 15		742.96	7.04		527.12	91.32		1.41	0.08		80.43	106.19
Lower	58441.4	PF 16		791.18	8.82		536.23	103.53		1.48	0.09		80.48	109.37
Lower	58252.8*	PF 1		50.00			113.82			0.44			67.37	
Lower	58252.8*	PF 2		100.00			167.09			0.60			72.69	
Lower	58252.8*	PF 3		150.00			210.71			0.71			75.56	
Lower	58252.8*	PF 4		200.00			249.58			0.80			78.03	
Lower	58252.8*	PF 5		250.00			285.13			0.88			79.41	
Lower	58252.8*	PF 6		300.00			318.44			0.94			80.59	
Lower	58252.8*	PF 7		350.00			349.77			1.00			81.39	
Lower	58252.8*	PF 8		400.00			379.63			1.05			82.09	
Lower	58252.8*	PF 9		450.00			407.95			1.10			82.75	
Lower	58252.8*	PF 10		499.99	0.01		435.17	0.59		1.15	0.02		83.12	6.00
Lower	58252.8*	PF 11		549.77	0.23		461.11	8.19		1.19	0.03		83.32	30.10
Lower	58252.8*	PF 12		599.03	0.97		485.81	20.99		1.23	0.05		83.52	48.41
Lower	58252.8*	PF 13		647.90	2.10		508.92	36.14		1.27	0.06		83.70	62.21
Lower	58252.8*	PF 14		696.68	3.32		530.79	55.29		1.31	0.06		83.87	85.22
Lower	58252.8*	PF 15	0.00	744.86	5.14	0.02	547.36	72.56	0.01	1.36	0.07	2.15	83.99	89.75
Lower	58252.8*	PF 16	0.04	793.46	6.50	0.89	556.73	82.72	0.05	1.43	0.08	13.94	84.07	92.32
Lower	58064.2*	PF 1		50.00			113.84			0.44			71.41	
Lower	58064.2*	PF 2		100.00			170.27			0.59			76.77	
Lower	58064.2*	PF 3		150.00			216.22			0.69			79.73	
Lower	58064.2*	PF 4		200.00			257.17			0.78			82.28	
Lower	58064.2*	PF 5		250.00			294.71			0.85			83.80	

Figure: Example RAS profile output table for a series of steady flow simulations.

8.3 Reservoir Hydraulics

A reservoir subdomain is discretized in the water quality model as shown in the [figure below](#)(see [page 24](#)). Laterally and longitudinally uniform vertical layers are assumed. Multiple tributary inflows are assumed to be able to enter any of the active reservoir layers. Similarly, outflows may also be taken from multiple layers and are not restricted to the layer corresponding to the outlet elevation. As opposed to reach subdomains, where users are required to input hydrodynamic forcing variables (e.g., flows, conveyance areas) at each cell face, hydrodynamic forcing for reservoirs consists of:

- reservoir seepage
- the sum of evaporation and precipitation
- an array of reservoir inflow and outflow magnitudes

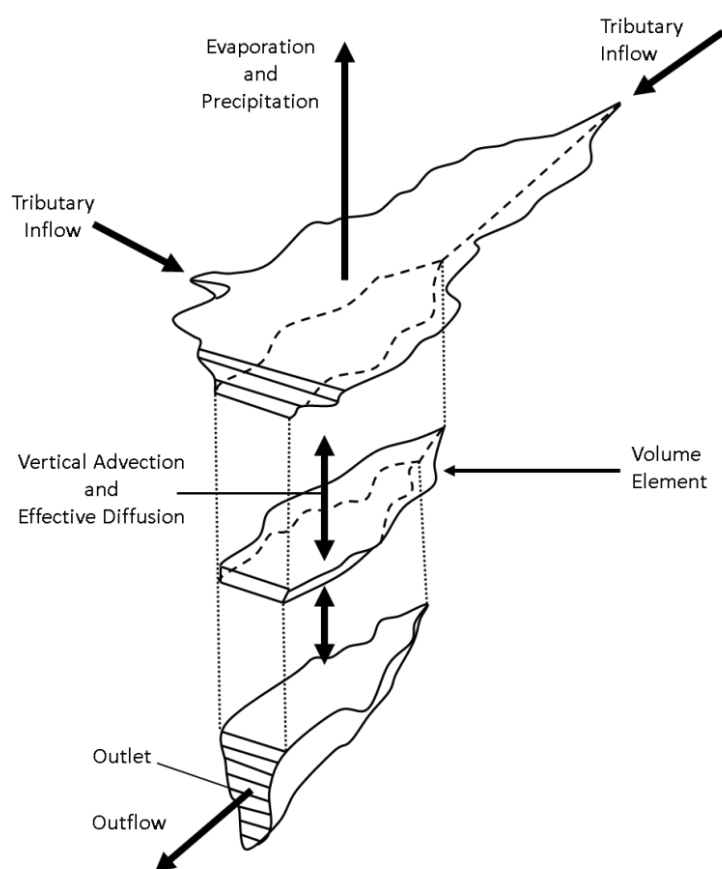


Figure: Discretization of a reservoir subdomain.

A compute step for the solution to the constituent transport equation ([Equation 1](#)([see page 0](#))) for a reservoir subdomain begins with the calculation of start-of-step and end-of-step layer volumes. Using this information, the minimum active top layer through the step is determined. An active top layer that becomes dry or becomes wet within a step is not considered for the allocation of inflows and outflows. Inflows and outflows are then distributed to each of the active layers using any stratification information available at the start of the step. If water temperatures are not modeled, a uniform distribution of inflow, weighted by the volume of the layer, is assumed. A similar procedure is used to allocate outflows.

Once all inflow and outflows are allocated to the layers, vertical velocities at the layer interfaces are calculated, using conservation of mass ([Equation 6](#)([see page 0](#))). This is done sequentially from the layer adjacent to the bed to the active surface layer. Effective vertical diffusion coefficients are then calculated, taking into account any information of reservoir stratification available. The number of subtime steps required for stability is calculated using [Equation 7](#)([see page 0](#)). For every sub-timestep, a new volume in the surface reservoir layer is calculated, advective and diffusive fluxes are calculated, concentrations are updated, and source-sink terms are calculated and applied. If the top layer volume falls below a pre-determined minimum volume, that layer is merged with the adjacent layer at the start of the time step loop. If the top layer volume exceeds the maximum volume for that layer (as determined by the elevation-storage curve data), that layer is split into a new active top layer. This is done at the end of the time step loop.

Following the completion of the time step loop, the vertical density profile is examined. If the density profile is not monotonically increasing from top to bottom of the reservoir (unstable stratification), layers are mixed downward until a stable velocity profile is achieved. A flow diagram of the compute process is given in the [figure below](#)([see page 25](#)).

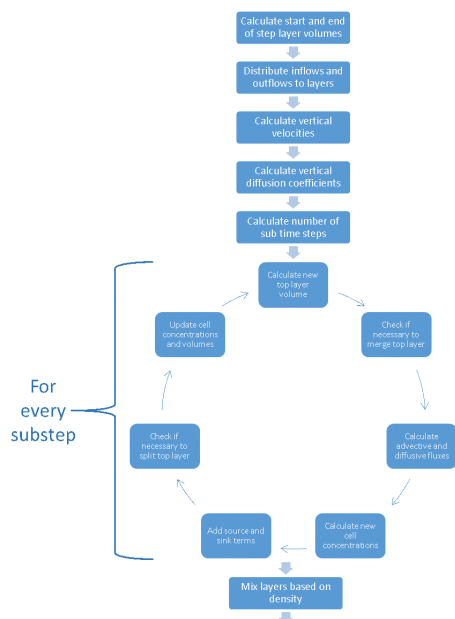


Figure: Reservoir constituent transport compute flow.

8.3.1 Equation of State

The equation of state used to calculate the density of water is given in Chapra (1996) as:

$$\rho = \rho_0 + AS + BS^{3/2} + CS^2 \quad \text{Equation 10}$$

where

ρ	= density of salt water (kg m ⁻³)
S	= salinity (ppt)
T	= temperature (°C)
A	= $8.24493 \times 10^{-1} - 4.0899 \times 10^{-3} T + 7.6438 \times 10^{-5} T^2 - 8.2467 \times 10^{-7} T^3 + 5.3875 \times 10^{-9} T^4$
B	= $-5.72466 \times 10^{-3} + 1.0227 \times 10^{-4} T - 1.6546 \times 10^{-6} T^2$
C	= 4.8314×10^{-4}

ρ_o	= density of fresh water
	= $999.842594 + 6.793952 \times 10^{-2}T - 9.095290 \times 10^{-3}T^2 + 1.001685 \times 10^{-4}T^3 - 1.120083 \times 10^{-6}T^4 + 6.536332 \times 10^{-9}T^5$

8.3.2 Inflow Allocation

The allocation of inflow volume to individual reservoir layers in the water quality prototype uses the same methods used in CE-QUAL-R1 (Environmental Laboratory, 1995) and HEC-5Q (HEC, 1986). First, the vertical layer with the density closest to the inflowing water density is identified; this is referred to as the insertion layer. Next, the extent of the upper and lower mixed zones (epilimnion and hypolimnion) are identified. If the insertion layer is within one of those zones, the inflow is partitioned uniformly across the mixed layer. If the insertion layer is within a zone of stratification, the thickness of the inflow zone (d) needs to be determined. This is calculated as a function of the state of stratification in the reservoir, the inflow rate, and the effective reservoir length (L).

$$d = 1.35 \left(\frac{QL}{A} \frac{1}{\sqrt{g \frac{\Delta\rho}{\rho_o}}} \right)^{2/3} \quad \text{Equation 11}$$

where

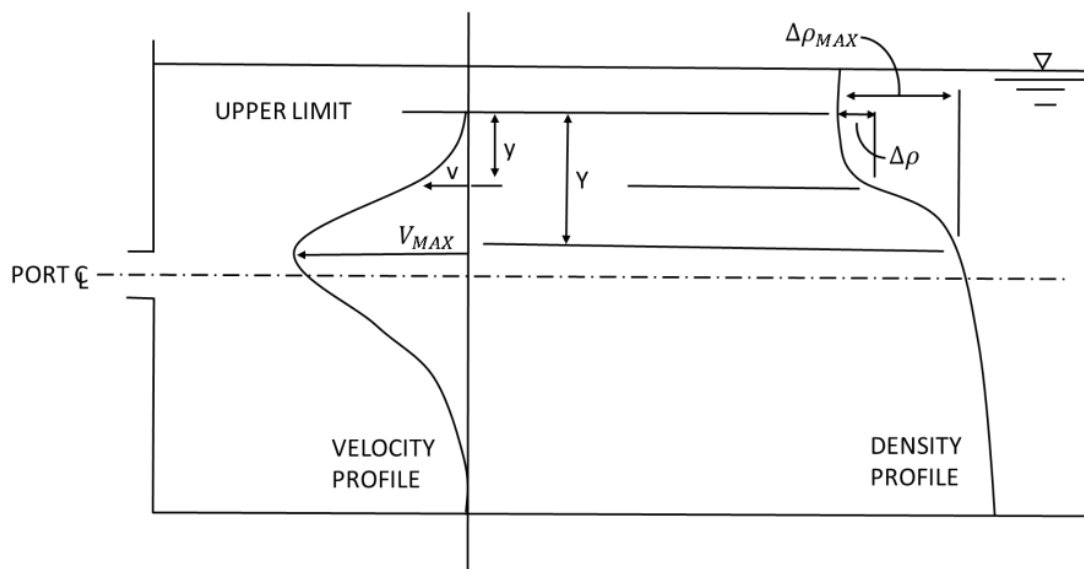
d	= one-half of the thickness of the inflow zone
Q	= inflow rate
L	= effective reservoir length
A	= lower horizontal surface area of the inflow layer
g	= acceleration due to gravity
$\Delta\rho$	= absolute value of the difference in density between the inflow layer and the upper or lower boundary layer of the inflow zone
ρ_o	= density of the inflow layer

Half-thickness lengths are calculated separately for the upper and lower inflow bounds. Because the density difference on the right-hand side of [Equation 11](#) (see page 26) depends on the inflow zone thickness, the solution must be calculated iteratively. This is accomplished in the water quality engine using the bisection method. Once the limits of the inflow region are determined, the inflow is partitioned uniformly across those layers. Multiple inflows are allocated to layers assuming superposition.

8.3.3 Withdraw Allocation

The allocation of outflow to individual reservoir layers in the water quality prototype also uses the same methods used in CE-QUAL-R1 (Environmental Laboratory, 1995) and HEC-5Q (HEC, 1986). The decision logic follows that

First, the upper and lower boundaries of the withdrawal are determined. A typical velocity profile for a withdrawal is shown in the [figure below](#)([see page 27](#)). If the withdrawal port elevation is located close to the water surface elevation or the reservoir bottom, the physical boundary will interfere with the velocity profile ([Figure](#)([see page 29](#))).



Whether or not interference exists at the upper or lower withdrawal boundaries is determined using [Equation 12](#)(see [page 27](#))

$$N = \sqrt{\frac{\Delta\rho}{\rho} \frac{g}{Z}} \quad \text{Equation 13}$$

Q	= outflow rate
Z	= distance between the upper or lower withdrawal boundary and the outlet centerline
Θ	= withdrawal angle in radians ($=\pi$ for withdrawals near the dam wall)
N	= buoyancy frequency
$\Delta\rho$	= absolute value of the difference in density between the outlet centerline and the upper or lower withdrawal boundary
ρ	= density at the outlet centerline

Interference is determined by substituting the distance from the outlet to the physical boundary for Z and the density different between the outlet layer and the surface or bottom layer for $\Delta\rho$. If the left-hand side of Equation 12(see page 27) is positive, then interference exists.

If the evaluation of the left-hand side of Equation 12(see page 27) is negative for both upper and lower boundaries, then the withdrawal zone forms freely. Equation 12(see page 27) is solved iteratively to determine the withdrawal limits. The distance from the lower boundary to the elevation of maximum velocity is then calculated

$$Y_L = H \left[\sin \left(1.57 \frac{Z_L}{H} \right) \right]^2 \quad \text{Equation 14}$$

where

Y_L	= distance between the lower boundary and the elevation of maximum velocity
H	= total thickness of the inflow zone
Z_L	= distance between the outlet centerline and the lower boundary elevation

As shown in the figure above(see page 27), this elevation is often above the outlet centerline. A normalized, parabolic velocity profile is then calculated for each layer in the inflow zone

$$v_i^{norm} = \left[1 - \frac{y_i}{Y} \frac{\Delta\rho_i}{\Delta\rho_{MAX}} \right]^2 \quad \text{Equation 15}$$

where

v_{normi}	= normalized outflow velocity for layer i
y_i	= distance between the elevation of maximum velocity and the elevation of the centerline of layer i
Y	= distance between the elevation of maximum velocity and the upper or lower withdrawal boundary
$\Delta\rho_i$	= absolute value of the density difference between the layer containing the maximum velocity elevation and layer i
$\Delta\rho_{MAX}$	= absolute value of the density difference between the layer containing the maximum velocity elevation and the layer containing the upper or lower withdrawal boundary

Finally, inflows to each layer (Q_i) are calculated as

$$Q_i = \frac{v_i^{norm}}{\sum_i v_i^{norm}} Q \quad \text{Equation 16}$$

where Q is the total inflow.

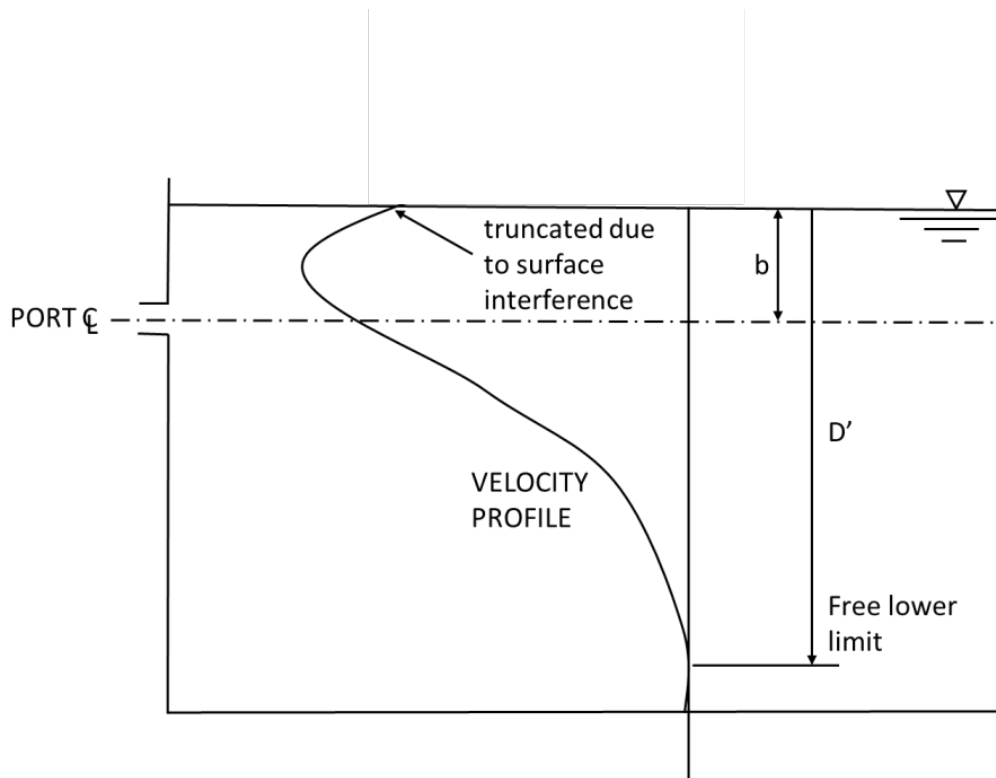


Figure: Reservoir withdrawal velocity profile with surface interference (redrawn from Davis et al., 1987).

If interference at the surface or bottom exists, the freely-forming withdrawal layer is calculated from

$$\frac{Q}{D'^3 N} - \frac{0.125\phi}{X^3} \frac{\theta}{\pi} = 0 \quad \text{Equation 17}$$

$$\phi = \frac{1}{2} \left[1 + \frac{1}{\pi} \sin \left(\frac{\frac{b}{D'}}{1 - \frac{b}{D'}} \pi \right) + \frac{\frac{b}{D'}}{1 - \frac{b}{D'}} \right] \quad \text{Equation 18}$$

$$X = \frac{1}{2} \left[1 + \frac{\frac{b}{D'}}{1 - \frac{b}{D'}} \right] \quad \text{Equation 19}$$

$$N = \sqrt{\frac{\Delta\rho}{\rho} \frac{g}{D'}} \quad \text{Equation 20}$$

where

b	= distance from the outlet elevation to the interference boundary
D'	= distance from the interference boundary to the freely-forming boundary
$\Delta\rho$	= absolute value of the density difference between the surface layer and lower free boundary (for surface interference) or between the bottom layer and upper free boundary (for bottom interference)

$\Delta\rho_i$	= absolute value of the density difference between the layer containing the maximum velocity elevation and layer i
----------------	--

A theoretical withdrawal boundary, located outside of the reservoir elevation limits, can be calculated for the boundary experiencing interference using [Equation 12](#)(see page 27). In this case, $\Delta\rho$ is extrapolated outside of the reservoir limits using linear interpolation. The calculation of the maximum velocity elevation is the same as given in [Equation 14](#)(see page 28), and the normalized velocities for the half of the outflow experiencing interference are calculated using [Equation 21](#)(see page 30), instead of [Equation 15](#)(see page 28).

$$v_i^{norm} = 1 - \left[\frac{y_i}{Y} \frac{\Delta\rho_i}{\Delta\rho_{MAX}} \right]^2 \quad \text{Equation 21}$$

For the case where both upper and lower boundaries experience interference, theoretical withdrawal limits are calculated for both. Multiple outflows are allocated to layers assuming superposition.

8.3.4 Effective Vertical Diffusion

The effective vertical diffusion coefficients (K_v) in the water quality engine are calculated using the same method used in HEC-5Q (HEC, 1986). A critical water column stability (or normalized density gradient), E_{crit} , is defined, where

$$E = \frac{1}{\rho} \frac{\partial \rho}{\partial z} \quad \text{Equation 22}$$

is the water column stability. Below this critical stability, the water is well-mixed and the effective diffusion coefficient is set at a constant maximum value. Above this critical stability, density stratification limits mixing, and the diffusion coefficients are calculated using a power law relationship.

$$K_v = \begin{cases} A_1 & \text{if } E \leq E_{crit} \\ A_2 E^{A_3} & \text{otherwise} \end{cases} \quad \text{Equation 23}$$

where

A_1	= maximum effective diffusion coefficient (range $1 \times 10^{-4} - 2 \times 10^{-4} \text{ m}^2 \text{ s}^{-1}$)
A_2	= empirical constant (4.6)
A_3	= empirical constant (0.7)

A density profile and the resulting effective diffusion coefficients for this method are shown in the [figure below](#)(see page 31).

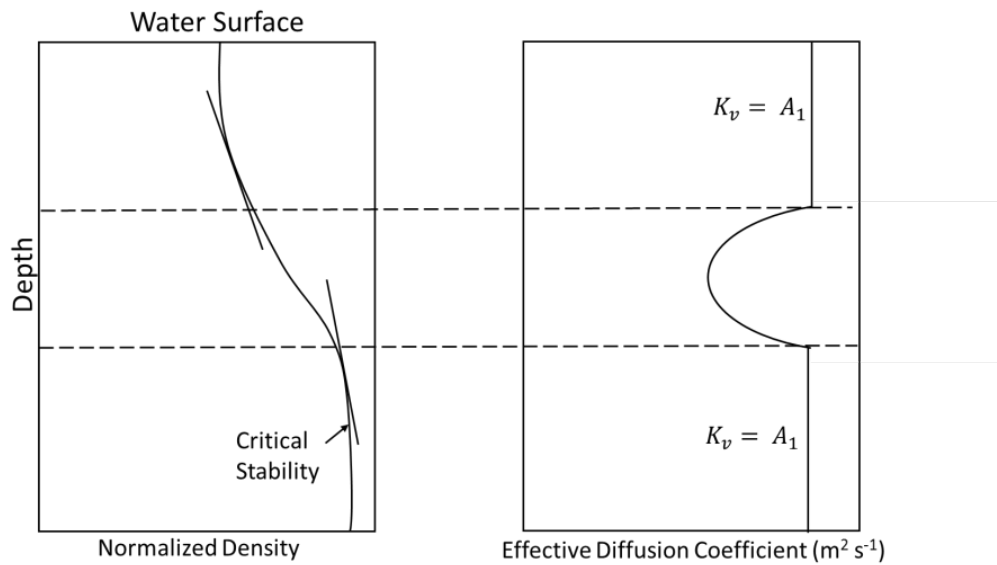


Figure: Reservoir effective vertical diffusion coefficient

8.3.5 Mixing

Separate logic is required in reservoir simulations in order to mix vertical layers and achieve stable stratification profiles. Stable stratification profiles are required for the correct functioning of the inflow and outflow allocation routines. Unstable profiles may result from convective cooling at the reservoir surface layer. In a physical reservoir, this instability would result in "penetrative convection" which would physically mix the layers. In the water quality engine, the same approach is taken as is used in HEC-5Q, whereby a filter is applied following the solution to the constituent transport equation. Starting from the water surface, the density of the current layer is checked against the density of the layer immediately below it. If an instability is present, the layers are mixed by redistributing the constituent masses to achieve equal concentrations in both layers. The process is repeated, mixing more total layers, until a stable density profile is achieved.

Additional sources of energy (other than penetrative convection) act to mix layers in a reservoir. These include (Fisher, et al.)

- mixing from momentum transfer of wind energy at the water surface
- inflow and outflow mixing
- mixing of internal layers due to seiching

Future development of the water quality engine may take these sources of mixing into account.

Two test applications are presented to verify the initial development of the water quality engine. The first application tracks a conservative scalar constituent through a ResSim network. The network includes a wide range of element types, connectivities, and features such as would be expected in an actual ResSim compute. The water quality simulation is checked for global conservation of mass. The second test application is the simulation of water temperature in a reservoir. Boundary conditions and meteorological data are obtained from a pre-existing HEC-5Q model of the reservoir. The simulation is checked for the seasonal development and breakdown of stratification, and modeled results are compared against observed vertical temperature profiles taken during the simulation period.

A ResSim network was created to include the wide range element types, connectivities, and features that would be expected in actual ResSim simulations. The [figure below](#)(see [page 32](#)) shows a map of the network.



The simulation includes the following features:

- reach and reservoir elements
- river confluences
- reservoirs with multiple inflows
- unsteady boundary inflows
- local inflows at river-river and river-reservoir junctions
- local diversions
- reservoir diverted outlets
- reservoir evaporation, precipitation, and seepage
- seasonal guide curve operation

The seasonal guide curve operations tests the water quality engine logic guiding the splitting/merging of vertical layers due to raising/dropping water levels. The [figure below](#)(see [page 33](#)) shows the unsteady flow timeseries used as the upstream boundary conditions for the simulation.

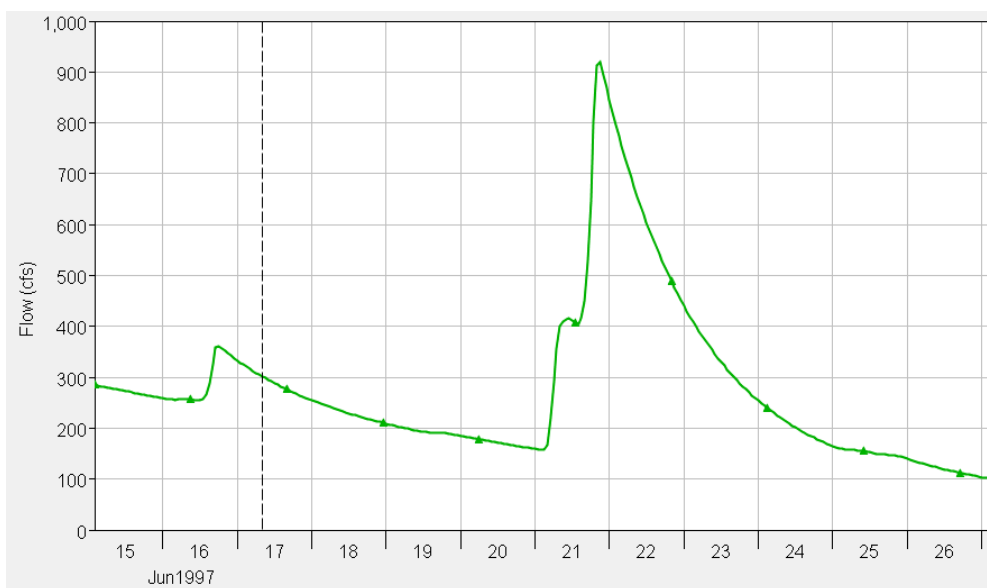


Figure: Upstream inflow time series for the conservative constituent routing test case.

An initial condition is specified where a cell in the middle of the upstream subdomain (Junction 7 to Junction 0; the northwestern most reach) and a cell in the middle of the Reservoir 1 subdomain (the farthest west reservoir) are prescribed concentrations equal to one. All other water quality cells are given initial concentrations of zero. The water quality simulation proceeds for the duration of the ResSim forecast period (~9 days).

Spatial plots of the routing of the tracer plumes through the ResSim network are shown in the [first figure below](#)(see [page 34](#)) and [second figure below](#)(see [page 35](#)). The plumes can be seen to be correctly transported through the network. The [third figure below](#)(see [page 36](#)) shows timeseries of the total scalar masses in each subdomain and the total mass for all subdomain. Tracer plumes are seen to enter and exit each subdomain. The total scalar mass in the system remains constant until the first tracer plume begins to exit the downstream boundary. This occurs around time step 140. Local diversion and diverted outlet flows were turned off for the simulation used to generate the conservation of mass plot.

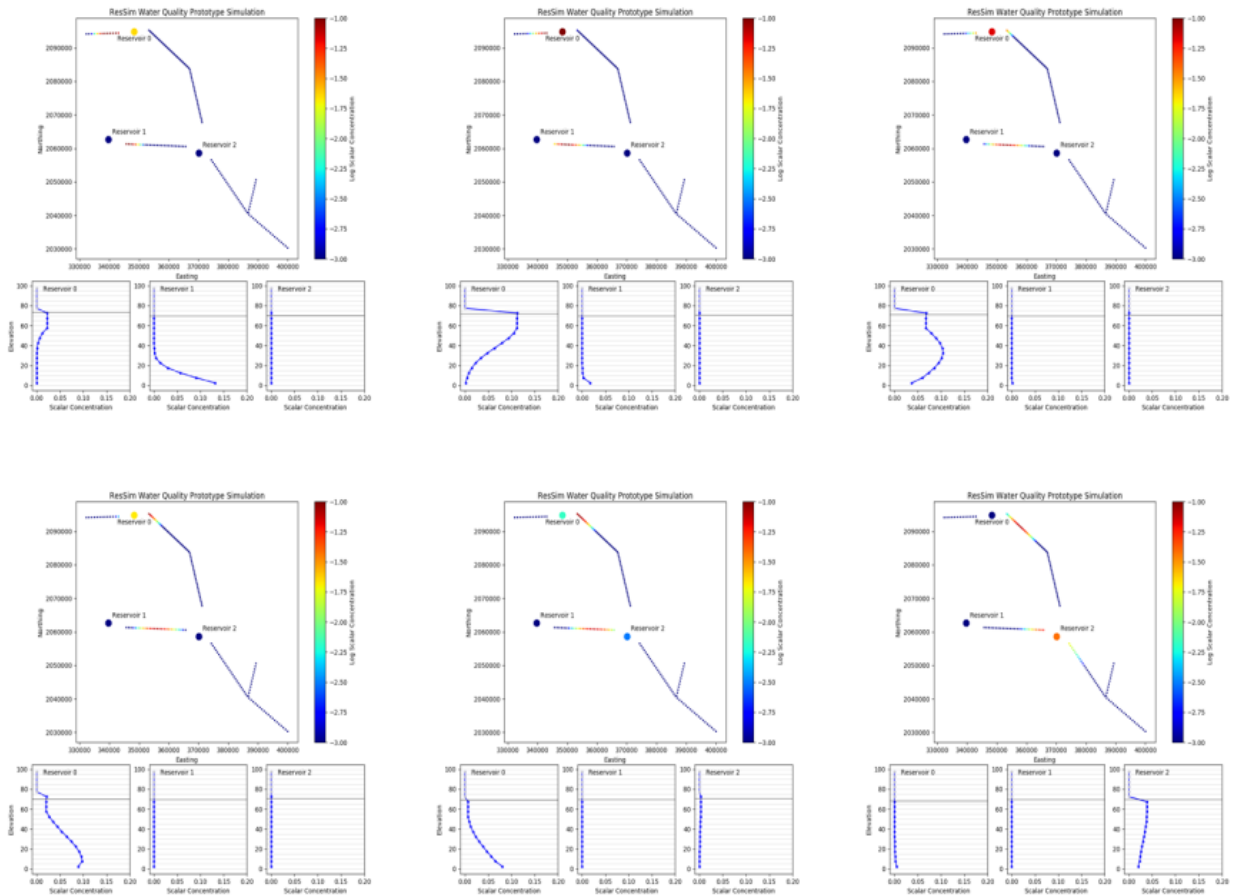


Figure: Routing of two plumes of a conservative constituent through a ResSim network. Panels show time steps from left to right, top to bottom. Scalar concentrations are shown for river water quality cells and the surface reservoir layer in the top subplot. Bottom subplots show vertical profiles in each reservoir. Plot 1 of 2.

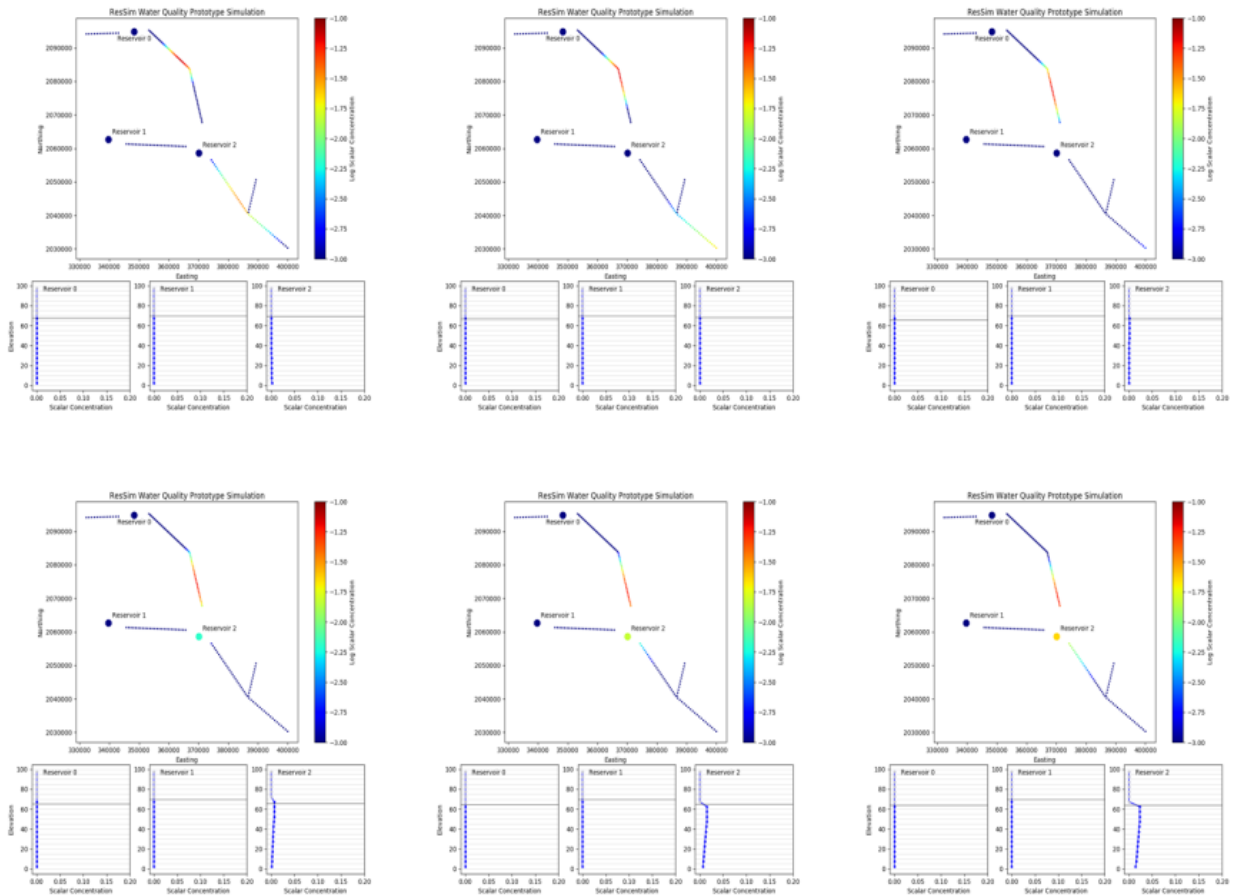


Figure Routing of two plumes of a conservative constituent through a ResSim network. Panels show time steps from left to right, top to bottom. Scalar concentrations are shown for river water quality cells and the surface reservoir layer in the top subplot. Bottom subplots show vertical profiles in each reservoir. Plot 2 of 2.

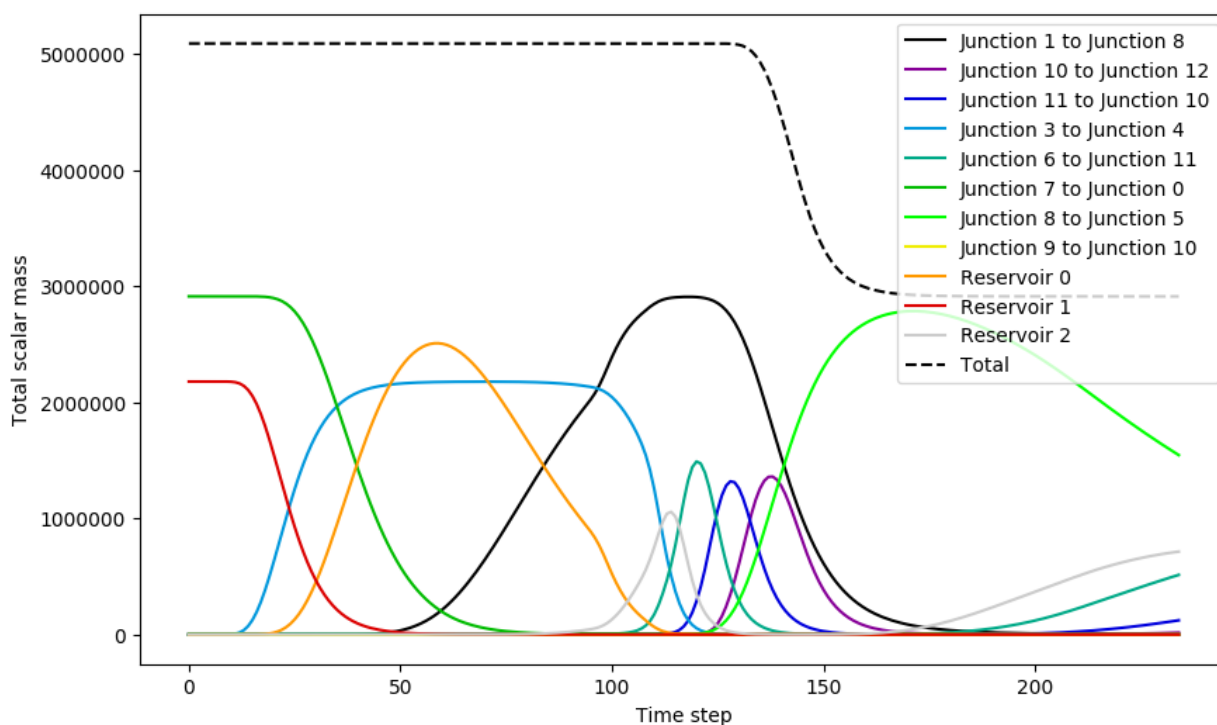


Figure: Timeseries of total scalar mass in each water quality subdomain for the conservative constituent routing test case.

9.2 Reservoir Water Temperature Simulation

A small ResSim network was created to simulate flows and water temperatures in Lake McClure, located on the Merced River in the San Joaquin Valley, California. This reservoir was chosen for an initial verification of the temperature modeling routines in the water quality engine because its vertical temperature profiles have previously been modeled successfully using HEC-5Q. Cleaned and verified boundary condition data for reservoir inflows, reservoir outflows, inflow water temperatures, and meteorological data were previously collected and available for use with the simulation. A large number of observed vertical temperature profiles were also available for comparison to model results.

A three year simulation period was chosen, spanning 2001 through 2003, based on the availability of observed data and the lack of use of the reservoir spillway in reservoir releases during that period. Inflow rates and water temperatures are shown in the [first figure below](#)(see page 37). Meteorological forcing data is shown in the [second figure below](#)(see page 38). Cloud cover fractions, necessary input to the ERDC-EL heat flux routines, were calculated by comparing the observed shortwave solar radiation against those predicted by astronomical conditions, using the equation (Martin and McCutcheon, 1999)

$$\frac{q_{obs}}{q_{astro}} = 1 - 0.65C_L^2 \quad \text{Equation 24}$$

where

q_{obs}	= observed solar irradiance
q_{astro}	= solar irradiance predicted by astronomical forcing

C_L

= cloudiness

A constant initial temperature profile of 12.4°C was used, corresponding to an observed temperature profile taken on January 11, 2001. Some minimal tuning of parameters associated with the calculation of vertical diffusion, evaporative heat flux, and penetration of shortwave radiation was performed to achieve a better match of modeled to observed data.

Comparison plots of modeled and observed temperature profiles are shown in four figures at the bottom of this section. With minimal calibration, the model is seen to do an accurate job of predicting the vertical temperature structure, including the seasonal setup and breakdown of stratification.

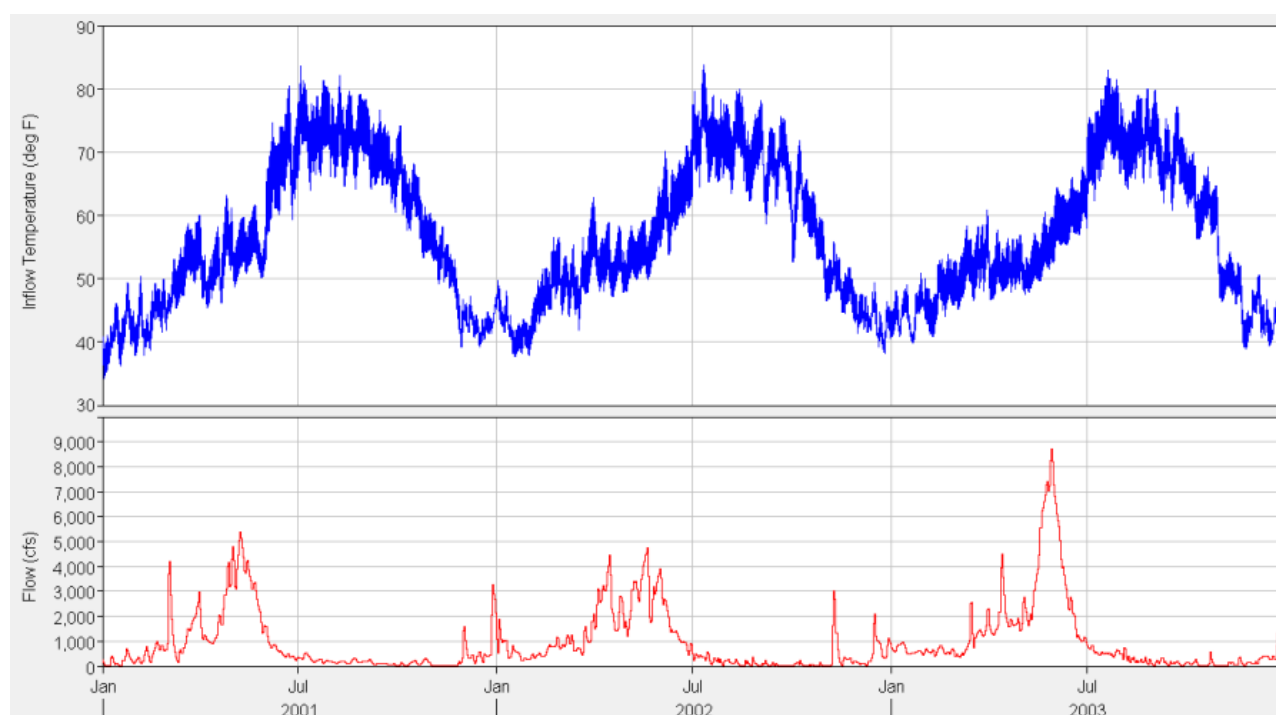


Figure: Lake McClure inflow rates and water temperatures, 2001–2003.

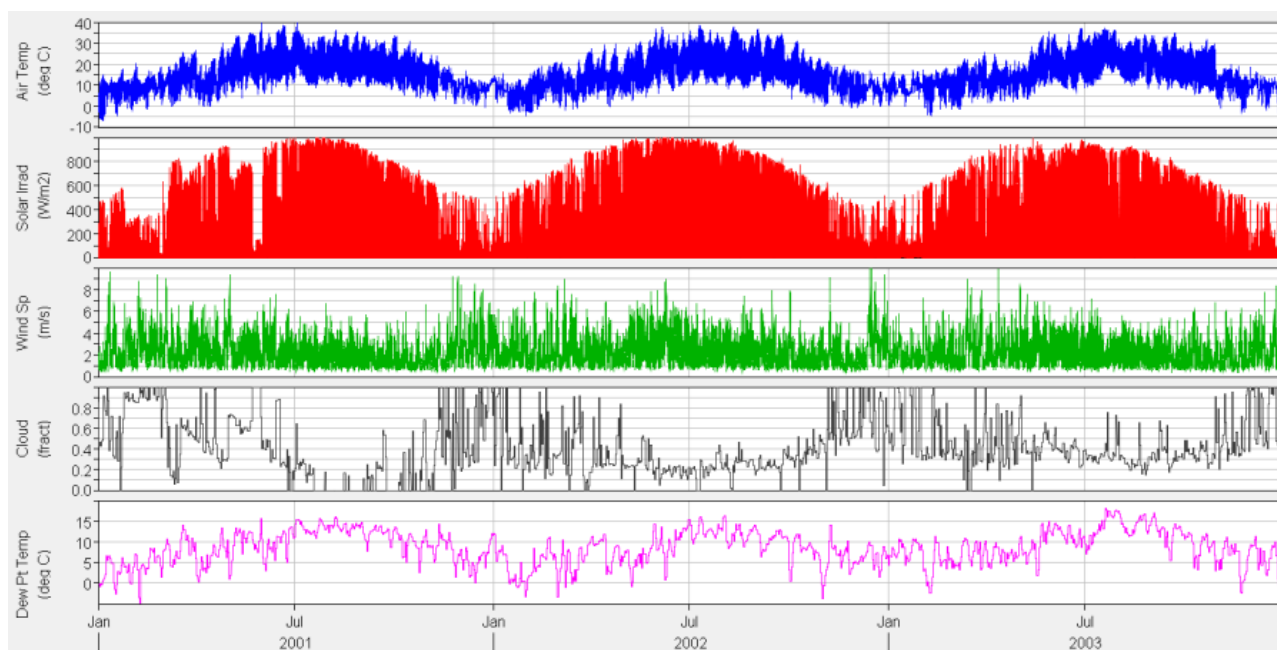


Figure: Lake McClure meteorological data, 2001–2003.

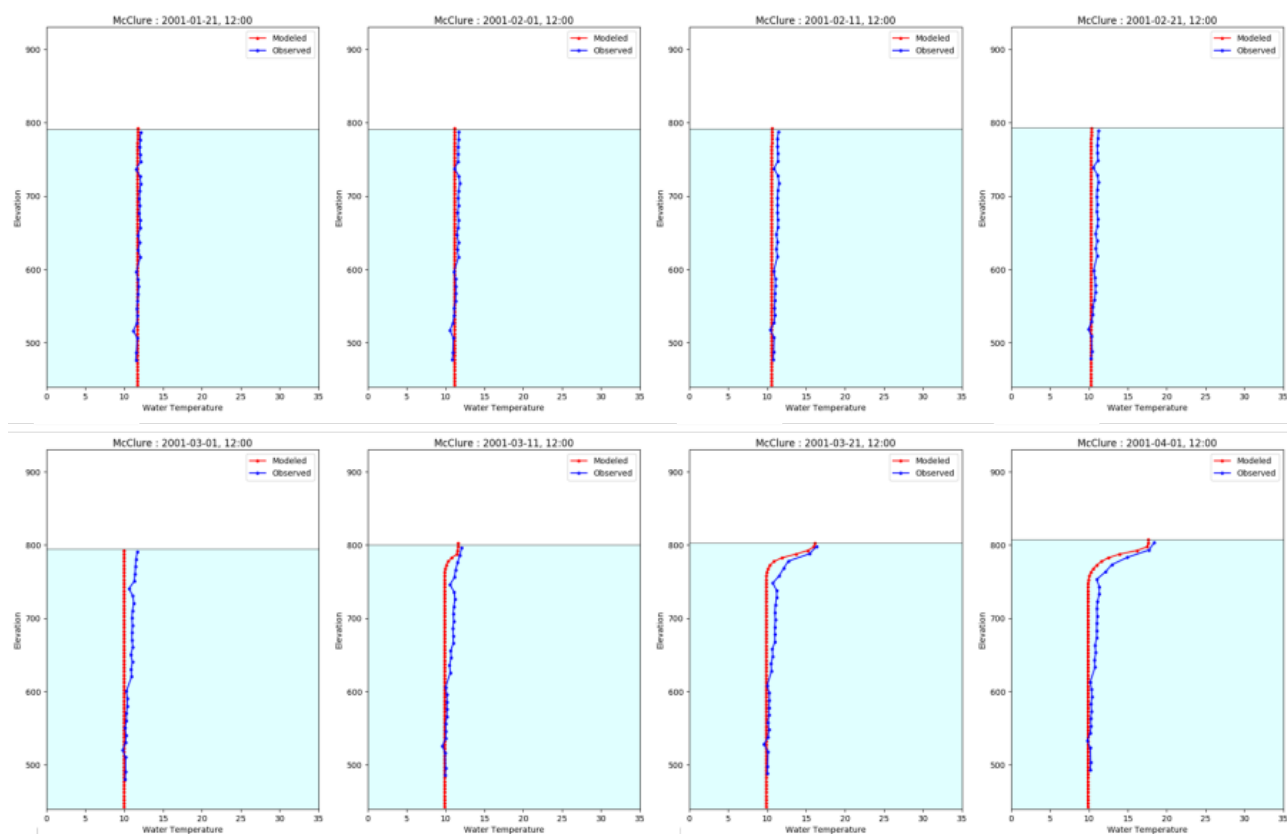


Figure: Lake McClure modeled and observed temperature profiles. Panels show snapshots when observed data profiles were taken, from left to right, top to bottom. Plot 1 of 4.

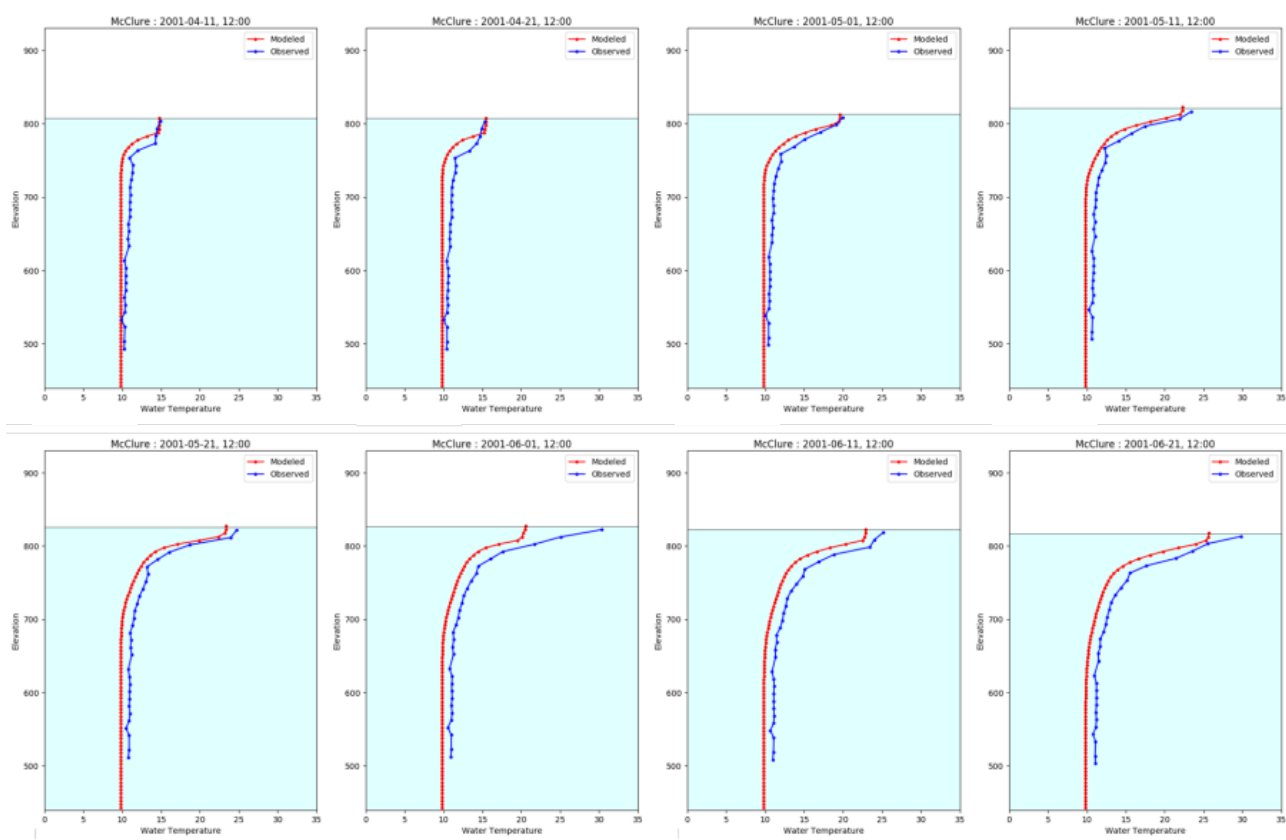


Figure: Lake McClure modeled and observed temperature profiles. Panels show snapshots when observed data profiles were taken, from left to right, top to bottom. Plot 2 of 4.

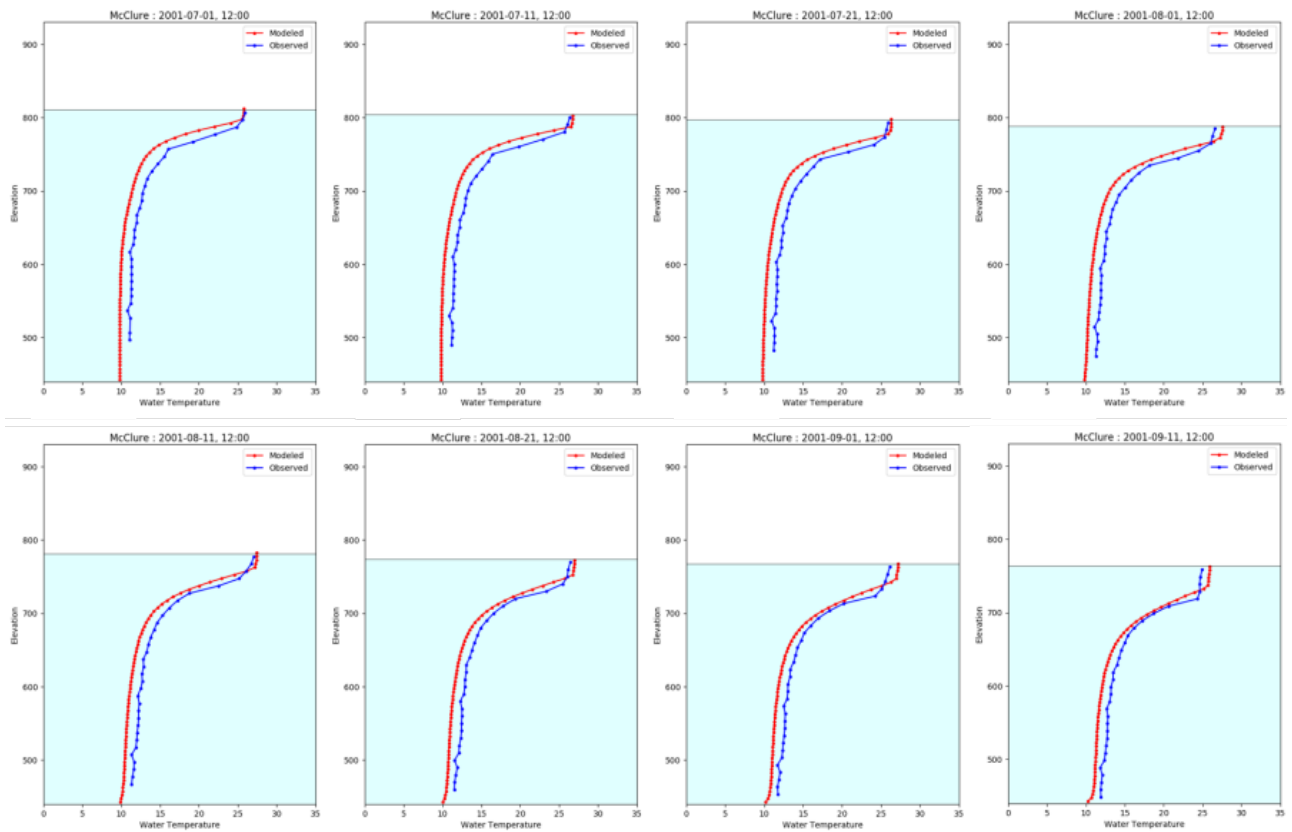


Figure: Lake McClure modeled and observed temperature profiles. Panels show snapshots when observed data profiles were taken, from left to right, top to bottom. Plot 3 of 4.

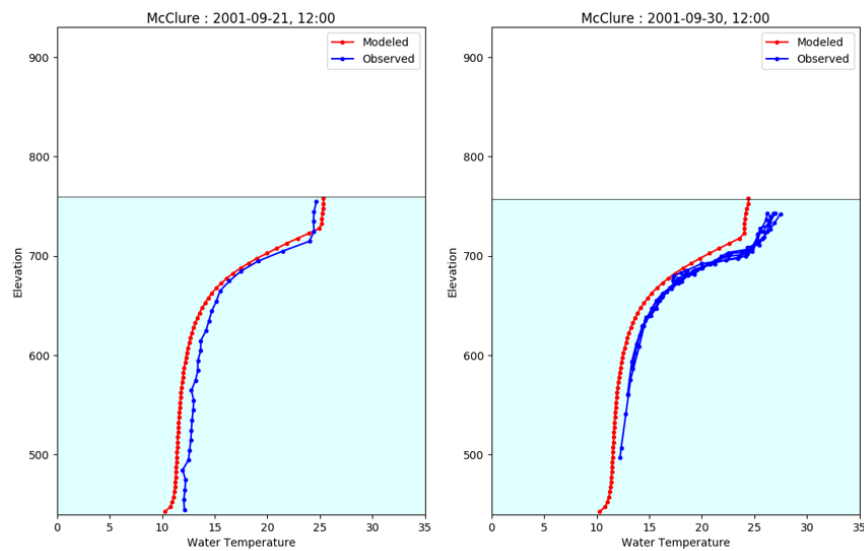


Figure: Lake McClure modeled and observed temperature profiles. Panels show snapshots when observed data profiles were taken, from left to right, top to bottom. Plot 4 of 4. Last subplot shows multiple profiles taken at different longitudinal locations within the lake.

10 References

Chapra, S.C. 1996. Surface water quality modeling. McGraw-Hill, Boston, MA.

Davis, J.E., Holland, J.P., Schneider, M.L., Wilhelms, S.C. 1987. SELECT, a numerical, one-dimensional model for selective withdrawal. Technical Report E-87-2 US Army Engineers Waterways Experiment Station. Vicksburg, MS.

Environmental Laboratory. 1995. "CE-QUAL-R1: A numerical one-dimensional model of reservoir water quality; User's manual." Instruction Report E-82-1 (Revised Edition), U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS.

Fischer, H.B., List, E.J., Koh, R.C.Y., Imberger, J., Brooks, N.H., 1979. Mixing in inland and coastal waters. Academic Press, New York.

[HEC] U.S. Army Engineer Hydrologic Engineering Center. 1986. "HEC-5 simulation of flood control and conservation systems: Appendix on water quality analysis." Report CPD-5, U.S. Army Engineer Hydrologic Engineering Center, Davis, CA.

Martin, J.L., McCutcheon S.C., 1999. Hydrodynamics and transport for water quality modeling. Lewis Publishers, New York.

11 Appendix A: Java API For WQEngineAdapter

11.1 Enumerated Types

RTNVAL

Defined values for success or failure of method in water quality compute

Constants:

SUCCESS (0)

FAIL (1)

OTHER (999)

STATE

Defined values for types of saved states

Constants:

INITIAL (0)

OUTER_LOOP (1)

INNER_LOOP (2)

11.2 Native Methods

jniInit(int unitSystem)

Initializes the WQEngine dll. Sets units system, opens log file.

Parameters:

unitSystem – int describing units system, from hecjavadevv4.0/code/hec/heclib/util/Unit.java

Returns:

RTNVAL.id – int describing success or failure of native method

jniSetNumWQSubdomains(int nSubdomains)

Sets the number of water quality subdomains

Parameters:

nSubdomains – number of water quality subdomains

Returns:

RTNVAL.id – int describing success or failure of native method

jniSetSubDomainName(int iwqsd, String sdname, int sdnameLength)

Sets the subdomain name

Parameters:

iwqsd – index of the water quality subdomain

sdname – subdomain name

sdnameLength – length of subdomain name string

Returns:

RTNVAL.id – describing success or failure of native method

jniSetSubDomainGeo(int iwqsd, int wqsdType, int numCells, int numFaces, int numFacesInternal, int[][] faceCells, int[][] cellFaces, int[] bcidx, double[] lngth)

Sets the subdomain's geometry

Parameters:

iwqsd – index of the water quality subdomain

wqsdType – int describing water quality subdomain type (WQGeoSubDomain.DOMAIN_TYPE)

numCells – number of cells in subdomain

numFaces – total number of faces in subdomain

numFacesInternal – number of internal faces in subdomain

faceCells – array of face indices for each cell

cellFaces – array of cell indices for each face

bcidx – indices of boundaries for each flow face

lngth – length or height of each cell

Returns:

RTNVAL.id – int describing success or failure of native method

jniSetResElevStorArea(int iwqsd, int nElevStor, double[][] resElevStor, int nElevArea, double[][] resElevArea)

Sets the reservoir elevation storage area data for a res subdomain

Parameters:

iwqsd – index of the water quality subdomain

nElevStor – number of elevation-storage points

resElevStor – array of reservoir elevation-storage data

nElevArea – number of elevation-area points

resElevArea – array of reservoir elevation-area data

Returns:

RTNVAL.id – int describing success or failure of native method

jniSetOutletData(int iwqsd, double area, double elev)

Sets the outlet area and elevation for a reservoir subdomain

Parameters:

iwqsd – index of the water quality subdomain

area – cross-sectional area of the outlet

elev – elevation of the outlet

Returns:

RTNVAL.id – int describing success or failure of native method

[jniSetNumWQSubdomainBoundaries\(int nSubdomainBoundaries\)](#)

Sets the number of water quality subdomain boundaries

Parameters:

nSubdomainBoundaries – number of subdomain boundaries

Returns:

RTNVAL.id – int describing success or failure of native method

[jniSetSubDomainBoundary\(int ibc, int ibcType, int\[\] connection\)](#)

Initializes a subdomain boundary connecting two subdomains

Parameters:

ibc – index of the subdomain boundary

ibcType – int describing the type of boundary (WQGeoSubDomainBoundary.BOUNDARY_TYPE)

connection – array of length 2 with upstream and downstream subdomain indices

Returns:

RTNVAL.id – int describing success or failure of native method

[jniSetRunTimeInfo\(String startTime, String endTime, double deltaT\)](#)

Sets simulation time window and time step

Parameters:

startTime – simulation start time string (Format: '2017-01-25, 00:00')

endTime – simulation end time string

deltaT – simulation time step (seconds)

Returns:

RTNVAL.id – int describing success or failure of native method

[jniSetNumWQConstituents\(int nWQconstituents\)](#)

Sets the number of water quality constituents

Parameters:

nWQconstituents – number of water quality constituents

Returns:

RTNVAL.id – int describing success or failure of native method

jniSetWQConstituentName(int iwqc, String wqcName, int wqcNameLength)

Sets the water quality constituent name

Parameters:

iwqc – index of the water quality constituent

wqcName – water quality constituent name (e.g., 'Temperature')

wqcNameLength – length of constituent name string

Returns:

RTNVAL.id – int describing success or failure of native method

jniSetBoundaryConcTS(int bcIdx, int wqConstitIdx, double[] vals)

Sets the time series of values for a constituent on an inflow boundary

Parameters:

bcIdx – index of the boundary

wqConstitIdx – index of the water quality constituent

vals – array of boundary values n time steps + 1 (start and end of step)

Returns:

RTNVAL.id – int describing success or failure of native method

jniSetNumMetStations(int nMetStations)

Sets the number of meteorological stations

Parameters:

nMetStations – number of met stations

Returns:

RTNVAL.id – int describing success or failure of native method

jniSetMetData(int iMetStat, int nSteps, double[][] metaData)

Sets the meteorological data for a met station

Parameters:

iMetStat – index of the met station

nSteps – number of values in the time window (forecast steps + 1)

metaData – array of met data (solar rad, wind sp, air temp, cloudiness, vapor pres)

Returns:

RTNVAL.id – int describing success or failure of native method

jniCompute(int startStep, int numSteps)

Instructs the engine to compute forward numSteps, beginning with startStep

Parameters:

startStep – starting step number

numSteps – total number of steps to compute

Returns:

RTNVAL.id – int describing success or failure of native method

jniSaveState(int state)

Saves the current state of the system for reloading at a later time

Parameters:

state – int describing type of state that is saved (this.STATE)

Returns:

RTNVAL.id – int describing success or failure of native method

jniRestoreState(int state)

Restores the state of the system to a previously saved value

Parameters:

state – int describing type of state that is saved (this.STATE)

Returns:

RTNVAL.id – int describing success or failure of native method

jniUpdateReachHydro(int iwqsd, double[] flow, double[] area, double[] sfcArea, int hydroUpdateType)

Updates the hydrodynamic data for a reach subdomain

Parameters:

iwqsd – index of the water quality subdomain

flow – array of flow values for every face in the subdomain

area – array of flow cross-sectional area for every face

sfcArea – array of surface area for every cell in the subdomain

hydroUpdateType – flag describing whether to update the start of step, end of step hydro, or both (WQTime.TIME_STEP_INFO)

Returns:

RTNVAL.id – int describing success or failure of native method

jniUpdateResHydro(int sldidx, double stor, double seep, double evapPrecip, double[] flows, int hydroUpdateType)

Updates the hydrodynamic data for a reservoir subdomain

Parameters:

sldIdx – index of the water quality subdomain

stor – reservoir storage

seep – reservoir seepage flow

evapPrecip – reservoir evaporation + precipitation flow

flows – array of reservoir inflows and outflows

hydroUpdateType – flag describing whether to update the start of step, end of stephydro, or both (WQTime.TIME_STEP_INFO)

Returns:

RTNVAL.id – int describing success or failure of native method

jniSetInitialCellVolumes()

Sets the cell volumes at the start of a simulation

Returns:

RTNVAL.id – int describing success or failure of native method

jniFinalizeCompute()

Finalizes the compute. Closes files, deallocates variables

Returns:

RTNVAL.id – int describing success or failure of native method

jniSetWQConstituentValAll(int wqConstitIdx, double val)

Sets a value for all cells in the domain for a given water quality constituent

Parameters:

wqConstitIdx – index of the water quality constituent

val – value

Returns:

RTNVAL.id – int describing success or failure of native method

jniSetWQConstituentValSubDomain(int wqConstitIdx, int subdomIdx, double val)

Sets a value for all cells in the subdomain for a given water quality constituent

Parameters:

wqConstitIdx – index of the water quality constituent

subdomIdx – index of the water quality subdomain

val – value

Returns:

RTNVAL.id – int describing success or failure of native method

jniSetWQConstituentValCell(int wqConstitIdx, int subdomIdx, int cellIdx, double val)

Sets a value for a particular cell in the subdomain for a given water quality constituent

Parameters:

wqConstitIdx – index of the water quality constituent

subdomIdx – index of the water quality subdomain

cellIdx – index of the cell

val – value

Returns:

RTNVAL.id – int describing success or failure of native method

[jniGetWQConstituentVal\(int subdomIdx, int wqConstitIdx, int paramType, int startEndFlag, int numCells\)](#)

Gets the concentration values for a given subdomain and water quality constituent

Parameters:

subdomIdx – index of the water quality subdomain

wqConstitIdx – index of the water quality constituent

paramType – int describing parameter to get (e.g., conc, mass) (WQIO.PARAM_TYPE)

startEndFlag – flag describing whether to get values from the start or end of step (WQTime.TIME_STEP_INFO)

numCells – number of cells in the water quality subdomain

Returns:

array of constituent param values

[jniGetHydroVal\(int subdomIdx, int paramType, int startEndFlag, int numCellsOrFaces\)](#)

Gets the hydro values used in the engine for a given subdomain

Parameters:

subdomIdx – index of the water quality subdomain

paramType – int describing parameter to get (e.g., volume, velocity) (WQIO.PARAM_TYPE)

startEndFlag – flag describing whether to get values from the start or end of step (WQTime.TIME_STEP_INFO)

numCellsOrFaces – number of cells or faces in the water quality subdomain (depends on paramType)

Returns:

array of hydro param values

11.3 Java Methods

[initCompute\(int unitSystem\)](#)

Loads the WQEngine dll. Sets units system, opens log file.

Parameters:

unitSystem – int describing units system, from hecjavadenv4.0/code/heclib/util/Unit.java

Returns:

RTNVAL success or failure of native method

[*setGeoData\(WQGeometry\(see page 42\) geo\)*](#)

Initializes the geometry data in the WQEngine for all subdomains

Parameters:

geo – object with list of subdomains

Returns:

RTNVAL – success or failure of native method

[*setBoundaryGeoData\(WQGeometry geo\)*](#)

Initializes the geometry data in the WQEngine for all subdomain boundaries

Parameters:

geo – object with list of subdomain boundaries

Returns:

RTNVAL – success or failure of native method

[*setRunTimeInfo\(WQTime wqtime\)*](#)

Initializes the simulation start/end time and time step in the WQEngine

Parameters:

wqtime – WQTime object containing simulation time info

Returns:

RTNVAL – success or failure of native method

[*setWQConstituentData\(WQConstituentSet wqConstitSet\)*](#)

Initializes the WQ constituent data in the WQEngine

Parameters:

wqConstitSet – set of WQConstituent objects

Returns:

RTNVAL – success or failure of native method

[*setBoundaryConcentrations\(WQGeometry\(see page 42\) geo\)*](#)

Initializes external (inflow) boundary concentration time series in the WQEngine

Parameters:

WQGeometry – object with list of boundaries containing concentration time series

Returns:

RTNVAL – success or failure of native method

setMetData(WQMetStationSet(see page 42) metStationSet)

Initializes met data time series in the WQEngine

Parameters:

metStationSet – set of WQMetStation objects containing met timeseries info

Returns:

RTNVAL – success or failure of native method

compute(int startStep, int numSteps)

Instructs the WQEngine to compute forward numSteps beginning with step startStep

Parameters:

startStep – starting time step

numSteps – number of steps to compute forward

Returns:

RTNVAL – success or failure of native method

saveState(STATE(see page 42) state)

Instructs the WQEngine to save concentrations at all WQ cells for loading later

Parameters:

state – int describing type of state that is saved (this.STATE)

Returns:

RTNVAL – success or failure of native method

restoreState(STATE(see page 42) state)

Instructs the WQEngine to load previously saved concentrations at all WQ cells

Parameters:

state – int describing type of state that is saved (this.STATE)

Returns:

RTNVAL – success or failure of native method

updateHydro(WQGeometry(see page 42) geo, int hydroUpdateType)

Updates the hydrodynamic data for all subdomains

Parameters:

geo – WQGeometry object with list of subdomains containing hydro info at a particular step

hydroUpdateType – flag describing whether to update the start of step, end of step hydro, or both (WQTime.TIME_STEP_INFO)

Returns:

RTNVAL – success or failure of native method

setInitialCellVolumes()

Initialize cell volumes for all cells in the WQ domain

Returns:

RTNVAL – success or failure of native method

finalizeCompute()

Finalizes the compute. Closes files, deallocates variables

Returns:

RTNVAL – success or failure of native method

setParam(int wqConstitIdx, double val)

Sets concentrations for a WQConstituent for all cells in the domain

Parameters:

wqConstitIdx – index of the WQ constituent

val – concentration value

Returns:

RTNVAL – success or failure of native method

setParam(int wqConstitIdx, int subdomIdx, double val)

Sets concentrations for a WQConstituent for all cells in the subdomain

Parameters:

wqConstitIdx – index of the water quality constituent

subdomIdx – index of the water quality subdomain

val – concentration value

Returns:

RTNVAL – success or failure of native method

setParam(int wqConstitIdx, int subdomIdx, int cellIdx, double val)

Sets concentrations for a WQConstituent for a cell in the subdomain

Parameters:

wqConstitIdx – index of the water quality constituent

subdomIdx – index of the water quality subdomain

cellIdx – index of the water quality cell

val – concentration value

Returns:

RTNVAL – success or failure of native method

getParam(int subdomIdx, int wqConstitIdx, int paramType, int startEndFlag, int numCells)

Gets the parameter values for a given subdomain and water quality constituent parameter

Parameters:

subdomIdx – index of the water quality subdomain

wqConstitIdx – index of the water quality constituent

paramType – int describing parameter to get (e.g., conc, mass) (WQIO.PARAM_TYPE)

startEndFlag – flag describing whether to get values from the start or end of step (WQTime.TIME_STEP_INFO)

numCells – number of cells in the subdomain

Returns:

RTNVAL – success or failure of native method

getHydroParam(int subdomIdx, int paramType, int startEndFlag, int numCellsOrFaces)

Gets the parameter values for a given subdomain and hydro parameter

Parameters:

subdomIdx – index of the water quality subdomain

paramType – int describing parameter to get (e.g., volume, velocity) (WQIO.PARAM_TYPE)

startEndFlag – flag describing whether to get values from the start or end of step (WQTime.TIME_STEP_INFO)

numCellsOrFaces – number of cells or faces (dependent on the parameter) in the subdomain

Returns:

RTNVAL – success or failure of native method

12 Appendix B: Fortran API

public subroutine initialize_compute(unit_sys, irtn) bind(c)

Opens log and debug output files. Sets unit system.

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	unit_sys	input unit system
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_nof_wq_subdomains(nwqsd, irtn) bind(c)

Sets the total number of water quality subdomains

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	nwqsd	total number of water quality subdomains
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_subdomain_name(iwqsd_in, sd_name, sd_name_len, irtn) bind(c)

Sets a subdomain name

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	iwqsd_in	index of water quality subdomain from Java
character(kind=c_char),	intent(in)			::	sd_name(*)	subdomain name
integer(kind=c_int),	intent(in)			::	sd_name_len	subdomain name length
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_subdomain_geometry(iwqsd_in, sd_type, ncells, nfaces, nfaces_internal, face_cells,
cell_faces, bc_idx, length, irtn) bind(c)

Sets water quality subdomain geometry information

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			:	iwqsd_in	index of water quality subdomain from Java
integer(kind=c_int),	intent(in)			:	sd_type	subdomain type (reach or reservoir)
integer(kind=c_int),	intent(in)			:	ncells	total number of water quality cells
integer(kind=c_int),	intent(in)			:	nfaces	total number of water quality cell faces
integer(kind=c_int),	intent(in)			:	nfaces_internal	number of internal faces
integer(kind=c_int),	intent(in)			:	face_cells	array of cells for each face
integer(kind=c_int),	intent(in)			:	cell_faces	array of faces for each cell
integer(kind=c_int),	intent(in)			:	bc_idx (nfaces-nfaces_internal)	boundary condition indexes for each flow face
real(kind=dp),	intent(in)			:	length (ncells)	array of cell lengths
integer(kind=c_int),	intent(out)			:	irtn	return flag indicating success of method

public subroutine set_reservoir_elev_stor_area(iwqsd_in, n_elev_stor, elev_stor, n_elev_area, elev_area, irtn)
bind(c)

Sets the reservoir elevation-storage and elevation-area data

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	iwqsd_in	index of water quality subdomain from Java
integer(kind=c_int),	intent(in)			::	n_elev_stor	number of elevation-storage data pairs
real(kind=dp),	intent(in)			::	elev_stor (n_elev_stor*2)	elev-storage data
integer(kind=c_int),	intent(in)			::	n_elev_area	number of elevation-area data pairs
real(kind=dp),	intent(in)			::	elev_area (n_elev_area*2)	elev-area data
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_outlet_data(iwqsd_in, area, elev, irtn) bind(c)

Sets area and elevation of a reservoir outlet

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	iwqsd_in	index of water quality subdomain from Java
real(kind=dp),	intent(in)			::	area	outlet cross-sectional area
real(kind=dp),	intent(in)			::	elev	outlet centerline elevation
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_nof_subdomain_boundaries(nsdb, irtn) bind(c)

Sets the number of water quality subdomain boundaries

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	nsdb	total number of subdomain boundaries
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_boundary(IBC_in, b_type, connection_idx, irtn) bind(c)

Sets boundary information

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	IBC_in	index of subdomain boundary from Java
integer(kind=c_int),	intent(in)			::	b_type	boundary type
integer(kind=c_int),	intent(in)			::	connection_idx(2)	subdomain connection indexes
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_run_time_info(start_time_str, end_time_str, hydro_dt, irtn) bind(c)

Sets the start and end times and hydro time step. Initializes current time.

Arguments

Type	Intent	Optional	Attributes		Name	
character(kind=c_char),	intent(in)			::	start_time_str(*)	start time string from C++
character(kind=c_char),	intent(in)			::	end_time_str(*)	end time string from C++
real(kind=c_double),	intent(in)			::	hydro_dt	hydro time step in seconds
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_nof_wq_constituents(nwqc, irtn) bind(c)

Sets number of water quality constituents and allocates arrays

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	nwqc	total number of water quality constituents
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_wq_constituent_name(iwqc_in, constit_name, constit_name_len, irtn) bind(c)

Sets wq constituent name

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	iwqc_in	index of water quality constituent from Java
character(kind=c_char),	intent(in)			::	constit_name (*)	constituent name
integer(kind=c_int),	intent(in)			::	constit_name_len	constituent name length
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_boundary_concentration_ts(IBC_in, iwqc_in, vals, irtn) bind(c)

Sets boundary concentration for a given subdomain and wq constituent idx, for all times in the simulation

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	IBC_in	index of subdomain boundary from Java
integer(kind=c_int),	intent(in)			::	iwqc_in	index of water quality constituent
real(kind=c_double),	intent(in)			::	vals (total_steps+1)	boundary concentration values

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_nof_met_stations(nms, irtn) bind(c)

Sets number of met stations and allocate arrays

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	nms	total number of met stations
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_met_data(ims_in, nvals_in, vals, irtn) bind(c)

Sets met data for all times in the simulation

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			: :	ims_in	index of met station from Java
integer(kind=c_int),	intent(in)			: :	nvals_in	number of time steps passed in for each dataset
real(kind=c_double),	intent(in)			: :	vals (nvals_in*nof_met_vars)	array of met data time series
integer(kind=c_int),	intent(out)			: :	irtn	return flag indicating success of method

public subroutine compute(istep, nsteps, irtn) bind(c)

Computes advection-diffusion solver for a time window

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	istep	start time step
integer(kind=c_int),	intent(in)			::	nsteps	number of time steps to compute
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine save_state(istate, irtn) bind(c)

Save a snapshot of all concentration values in all subdomains

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	istate	index of state to save to
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine restore_state(istate, irtn) bind(c)

Restore all concentrations in the water quality domain to values saved in a previous snapshot

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	istate	index of saved state to restore to
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine update_reach_hydro(iwqsd_in, flow, area, sfc_width, hydro_update_type, irtn) bind(c)

Sets hydro quantities for a reach subdomain

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			:	iwqsd_in	index of water quality subdomain from Java

Type	Intent	Optional	Attributes		Name	
real(kind=c_double),	intent(in)			:	flow (wq_subdomain(iwqsd_in+1)%nof_faces)	flow at cell faces
real(kind=c_double),	intent(in)			:	area (wq_subdomain(iwqsd_in+1)%nof_faces)	cross-sectional area at cell faces
real(kind=c_double),	intent(in)			:	sfc_width (wq_subdomain(iwqsd_in+1)%nof_faces)	surface width at cell faces
integer(kind=c_int),	intent(in)			:	hydro_update_type	flag indicating to update start or end of step
integer(kind=c_int),	intent(out)			:	irtn	return flag indicating success of method

public subroutine update_reservoir_hydro(iwqsd_in, storage, seepage, evap_precip, flows, hydro_update_type, irtn) bind(c)

Sets hydro quantities for a reservoir subdomain

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			:	iwqsd_in	index of water quality subdomain from Java
real(kind=c_double),	intent(in)			:	storage	reservoir storage
real(kind=c_double),	intent(in)			:	seepage	reservoir seepage flow
real(kind=c_double),	intent(in)			:	evap_precip	reservoir evaporation + precipitation flow

Type	Intent	Optional	Attributes		Name	
real(kind=c_double),	intent(in)			:	flows (wq_subdomain(iwqsd_in+1)%nof_boundary_faces)	array of reservoir inflows and outflows
integer(kind=c_int),	intent(in)			:	hydro_update_type	flag indicating to update start or end of step
integer(kind=c_int),	intent(out)			:	irtn	return flag indicating success of method

public subroutine set_initial_cell_volumes(irtn) bind(c)

Sets initial cell volumes for a subdomain

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine finalize_compute(irtn) bind(c)

Finalizes compute. Deallocates variables, closes files.

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_wq_constituent_val_all(iwqc_in, val, irtn) bind(c)

Sets a concentration for all cells in the wq domain

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	iwqc_in	index of water quality constituent from Java
real(kind=c_double),	intent(in)			::	val	concentration value

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_wq_constituent_val_subdomain(iwqc_in, iwqsd_in, val, irtn) bind(c)

Sets a concentration of a given cell to a particular value

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	iwqc_in	index of water quality constituent from Java
integer(kind=c_int),	intent(in)			::	iwqsd_in	index of water quality subdomain from Java
real(kind=c_double),	intent(in)			::	val	concentration value
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine set_wq_constituent_val_cell(iwqc_in, iwqsd_in, cell_idx_in, val, irtn) bind(c)

Sets a concentration of a given cell to a particular value

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	iwqc_in	index of water quality constituent from Java
integer(kind=c_int),	intent(in)			::	iwqsd_in	index of water quality subdomain from Java
integer(kind=c_int),	intent(in)			::	cell_idx_in	index of water quality cell from Java
real(kind=c_double),	intent(in)			::	val	concentration value
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine get_wq_constituent_val(iwqsd_in, iwqc_in, param_type, old_or_new_val, n_cells_in, vals, irtn)
bind(c)

Gets concentrations of all cells for a given subdomain and wq constituent idx

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			::	iwqsd_in	index of water quality subdomain from Java
integer(kind=c_int),	intent(in)			::	iwqc_in	index of water quality constituent from Java
integer(kind=c_int),	intent(in)			::	param_type	type of water quality constituent parameter
integer(kind=c_int),	intent(in)			::	old_or_new_val	flag to get values at start or end of step
integer(kind=c_int),	intent(in)			::	n_cells_in	number of cells from Java
real(kind=c_double),	intent(out)			::	vals (n_cells_in)	array of constituent parameter values
integer(kind=c_int),	intent(out)			::	irtn	return flag indicating success of method

public subroutine get_hydro_val(iwqsd_in, param_type, old_or_new_val, n_cells_or_faces_in, vals, irtn) bind(c)
Gets hydro parameter for all cells or faces for a given subdomain

Arguments

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			:	iwqsd_in	index of water quality subdomain from Java
integer(kind=c_int),	intent(in)			:	param_type	type of water quality constituent parameter
integer(kind=c_int),	intent(in)			:	old_or_new_val	flag to get values at start or end of step

Type	Intent	Optional	Attributes		Name	
integer(kind=c_int),	intent(in)			:	n_cells_or_faces_in	number of cells or faces from Java
real(kind=c_double),	intent(out)			:	vals	array of hydro parameter values
				:	(n_cells_or_faces_in)	
integer(kind=c_int),	intent(out)			:	irtn	return flag indicating success of method
				:		