***Core Pollutant Functions:***

* *See flowchart for comprehensive overview of nodes and links:* [*https://www.lucidchart.com/invitations/accept/319e57a8-ba2b-4886-af6a-30d6fda83719*](https://www.lucidchart.com/invitations/accept/319e57a8-ba2b-4886-af6a-30d6fda83719)
* *See flowchart for comprehensive overview of LIDs:* [*https://www.lucidchart.com/invitations/accept/d0471daf-39ed-4b14-8826-492996a84f70*](https://www.lucidchart.com/invitations/accept/d0471daf-39ed-4b14-8826-492996a84f70)

***Code Modification Options for Nodes & Links  
Pros & Cons:***

1. Setting removal R in the node following the methods created for link target\_setting
   1. Pros:
      1. Allows SWMM to route pollutants
      2. Allows SWMM to do other pollutant accumulations/mass balance
      3. No timestep issues
      4. Do not need setter for pollutant concentration
      5. Bypasses SWMM’s mathexpr\_eval() function which limits the math functions available
   2. Cons:
      1. Need to make R a global variable for both nodes and links
      2. getRemoval() called by treatmnt\_treat() checks if R between 0 and 1, so either need to write a different function for erosion or rewrite this function
2. Pollutant setter using the global variable newQual
   1. Pros:
      1. simple function to write
      2. can directly set pollutant concentration if you know it from treatment or sensor data
   2. Cons:
      1. Would need to build our own mass balance because will bypass SWMM’s internal mass balance
      2. Requires creating new global setter variables for both links and nodes
3. Bypass treatmnt\_treat() with an externally defined treatment function
   1. Pros:
      1. Uses global variables newQual
      2. Works for adding more pollutants to the network (like for erosion)
   2. Cons:
      1. It calls an error reporting function and mass balance equation, so we would need to make sure we still call that in our version
      2. Requires creating a new global variable for custom treatment for both links and nodes
4. Pollutant setter buts creates new global function externalQual.
   1. Pros:
      1. Simple to write – reuse existing code but switch newQual to externalQual
      2. allow you to directly set pollutant concentration if you know it (but could add a separate setter for that if that’s someone’s goal, for example, if they have pollutant data already for their network)
      3. Works for adding more pollutants to the network (like for erosion)
      4. allows for SWMM’s internal mass balance to occur
      5. Can replicate this method for LIDs
   2. Cons:
      1. Requires creating two new global variables for both links and nodes

***SWMM Code Modifications for Nodes & Links:***

* src/objects.h:
  + Lines 514-515: added the objects <int externalTreatment> and <double\* externalQual> for nodes
  + Lines 672-673: added the objects <int externalTreatment> and <double\* externalQual> for links
  + Abhi mentioned may want to do something like this to make a Boolean: <https://stackoverflow.com/questions/1921539/using-boolean-values-in-c>
* src/toolkitAPI.c:
  + Line 974: changed to get externalQual < result = Node[index].externalQual[pollutant\_index]; >
  + Line 1006: changed to set externalQual < Node[index].externalQual[pollutant\_index] = pollutant; >
  + Line 1007: added < Node[index].externalTreatment = 1; >
  + Line 1039: changed to get externalQual < result = Link[index].externalQual[pollutant\_index]; >
  + Line 1071: changed to set externalQual < Link[index].externalQual[pollutant\_index] = pollutant; >
  + Line 1072: added < Link[index].externalTreatment = 1; >
* src/treatmnt.c:
  + Line 290-343: added new function < void treatmnt\_custom(int j, double q, double v, double tStep) >
* include/toolkitAPI.h:
  + Line 119: added < SM\_EXTERNALTREATMENT\_N = 5, >
  + Line 131: added < SM\_EXTERNALTREATMENT\_L = 7, >
* src/project.c:
  + Line 1041: added <Node[j].externalQual = (double \*) calloc(Nobjects[POLLUT], sizeof(double));> to provide memory for the object
  + Line 1052: added <Link[j].externalQual = (double \*) calloc(Nobjects[POLLUT], sizeof(double));> to provide memory for the object
* src/funcs.h:
  + Line 238: added < void treatmnt\_custom(int node, double q, double v, double tStep); >
* src/qualrout.c:
  + Line 132-139: added < else { treatmnt\_custom(j, qIn, vAvg, tStep); } >

***PySWMM Code Modifications:***

* swmm5.py: (PySWMM and PySWMM\_Lite)
  + Lines 1657-1667: added function getNodePollutant() and setNodePollutant()
  + Lines 1669-1679: added function getLinkPollutant() and setLinkPollutant()
* environment.py: (PySWMM\_Lite only)
  + Line 210-211: created \_setLinkPollutant() function
  + Line 193-194: created \_getLinkPollutant() function
  + Line 181-182: created \_setNodePollutant() function
  + Line 178-179: created \_getNodePollutant() function
* toolkitapi.py: (PySWMM and PySWMM\_Lite)
  + Line 80: added <externalTreatment = 5> under <Class NodeParams>
  + Line 108: removed <seepRate = 7> and added <externalTreatment = 7> under <Class LinkParams>

***Testing Code Modifications:***

* Before I can test, need to download new SWMM C code:
  + cd into SWMM folder
  + type <rm -r CMakeFiles> enter
  + type <rm CMakeCache.txt> enter
  + type <cmake CMakeLists.txt
  + > enter
    - that remakes the Makefile which tells C compiler which files to compile
  + type <make> and enter
    - tells you what errors there are
  + go to lib folder, libswmm5.so is the file will all my SWMM functions
    - copy and paste libswmm5.so into PySWMM macros folder
* Test for Nodes:
  + Test one tank with constant inflow and constant pollutant load
    - Test the treatments that are available in SWMM
      * Exponential Removal
      * Percent Removal
      * Constant Effluent Conc.
  + Test one tank with variable inflow and variable pollutant load
    - Test the treatments that are available in SWMM
      * Exponential Removal
      * Percent Removal
      * Constant Effluent Conc.
  + Several tanks and only change 1 or 2 tanks with variable inflow and variable pollutant load
    - Test the treatments that are available in SWMM
      * Exponential Removal
      * Percent Removal
      * Constant Effluent Conc.
  + Compare SWMM vs PySWMM:
    - Concentration
    - Mass Balance
    - Make sure you check everything that can break
* Test for Links:
  + Do this once confirm it works for nodes
    - Create Basin to link to basin
    - Add pollutant to link
    - Do mass balance to confirm it makes sense

**SWMM Computational Steps (SWMM Manual)**

1. First the cumulative mass flow rate of each pollutant into each node of the network at the current time step is found. It includes pollutant loads from subcatchment runoff, dry weather sanitary flow, user-defined external time series loads, and possible groundwater and RDII flows, all evaluated at time *t*. To this is added the mass loads from all links (pipes, channels, pumps, etc.) that flow into the node. These are computed by multiplying the current outflow rate of the inflowing link (QL2(t+∆t)*)* by the link’s current pollutant concentration (cL(t)*).*
2. Then a new concentration is computed for each node in the network. If the node is a non- storage node, the concentration is simply the cumulative mass flow rate divided by the cumulative inflow rate (Equation 5-2 above). For a storage node, Equation 5-6 is used to compute a new mixture concentration cN(t+∆t) where Qin is the cumulative inflow rate from step 1 and Cin is step 1’s cumulative mass inflow divided by Qin.

A close up of a logo

Description automatically generated

1. Finally, Equation 5-6 is applied to determine a new concentration for each pollutant in each conduit, cL(t+∆t). In this equation, Qin is the flow rate sent into conduit from its upstream node, QL1(t+∆t)*,* and Cin is the newly updated concentration of this node, cN(t+∆t)*,* found in step 2. For links that have no volume (pumps, regulators, and dummy conduits) cL(t+∆t) is set equal to the upstream node concentration cN(t+∆t).

