SUSTAIN–Programmer's Manual: *Simulation Engine*

Submitted to

U.S. Environmental Protection Agency
Office of Research and Development
National Risk Management Research Laboratory
Water Supply and Water Resources Division
2890 Woodbridge Avenue (MS-104)
Edison, NJ 08837

Submitted by

Tetra Tech, Inc.
10306 Eaton Place, Suite 340
Fairfax, VA 22030

January 31, 2012

Contents

Tables

Figures

Class Index

File Index

SUSTAIN Software Design

The System for Urban Stormwater Treatment and Analysis INtegration (*SUSTAIN*) was developed by the U.S. Environmental Protection Agency (EPA) to support practitioners in developing cost-effective management plans for municipal stormwater programs and evaluating and selecting best management practices (BMPs) to achieve water resource goals. It includes a process-based, continuous-simulation BMP module for representing flow and pollutant transport routing through various types of structural BMPs. *SUSTAIN* simulates certain hydrologic and water quality processes that allow users to evaluate runoff volume reduction, peak flow attenuation, and pollutant load reduction. Users can select from various algorithms for certain processes depending on available data, consistency with coupled modeling assumptions, and the level of detail required. *SUSTAIN* includes a cost database composed of typical BMP component cost data from a number of published sources for reference purposes. Cost information for any BMP can be specified as a function of physical features such as excavation volume, soil media or other construction materials, or even aggregated cost per unit volume.

SUSTAIN extends the capabilities and functionality of traditionally available models by providing integrated analysis of water quantity, quality, and cost factors. The difference between *SUSTAIN* and other BMP models is *SUSTAIN*'s ability to evaluate a range of BMP locations, types, and sizes during stormwater management planning. Certain BMP properties in *SUSTAIN* are specifically represented as *decision variables*, meaning that they are permitted to change across a predefined range of variability during the course of model simulation to support BMP selection and placement optimization. As BMP size changes, so do cost and performance. *SUSTAIN* runs iteratively to generate a cost-effectiveness curve of BMP opportunities in the modeled study area to support decision making. Figure 1 shows *SUSTAIN*'s overall software design, including system components, relationships between components, and the general flow of information. Each of the components shown in Figure 1 are described in more detail in Chapter 2 of EPA-published *SUSTAIN* technical report (EPA/600/R-09/095) available at the EPA *SUSTAIN* website (http://www.epa.gov/nrmrl/wswrd/wq/models/sustain).

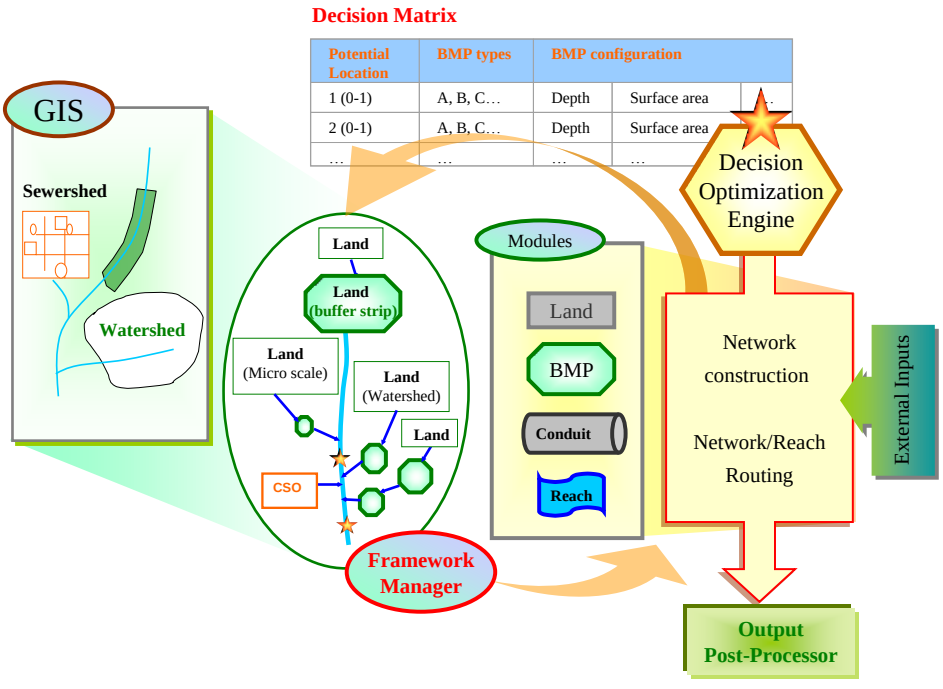


Figure 1. *SUSTAIN* components and flow chart.

SUSTAIN is designed to perform the following sequence of operations.

- From the GIS view (ArcMap 9.3.1) and database (File-based Geodatabase containing spatial and tabular dataset), the GIS interface is first used to develop a simulation network defining the relationship between land-area units, BMPs, and stream segments in a watershed.
- The purpose of the GIS interface is to facilite creation of a *SUSTAIN* model input file.
- *SUSTAIN* engine reads the input file and performs a run in which the external inputs (boundary conditions such as hydrology and water quality data timeseries and/or climate data timeseries) are passed to appropriate modules (i.e., land, BMP, conduit/reach) and their outputs are routed to receiving modules.
- The Optimization Engine evaluates the current option and selects the next preferred option in the Feasible Options Matrix (FOM) on the basis of cost and defined flow and water quality criteria. The preferred option can be a different combination of BMP locations and types as conceputualized in the *Decision Matrix* of Figure 1. The Decision Matrix is the user-specified range of decision variables (BMP configuration), but the Feasible Options Matrix is composed of feasible combinations of decision variables (types, configurations, locations, and costs) in context within the overall BMP network. While the true search space might include thousands, millions, or billions of possible combinations, depending on the number and ranges of decision variables, the purpose of the optimization algorithm is to navigate a more efficient path through the search space to reduce the number of searches by orders of magnitude. The FOM includes the range of solutions with different BMP configurations prepared by the optimization search techniques, taking into account the total cost and evaluation factors to compare against the target evaluation criteria. *SUSTAIN* performs numerous iterations until the user-defined convergence criteria are met.

Depending on how the land simulation is approached, *SUSTAIN* provides two options for defining loading conditions to BMPs: internal land simulation and external land simulation:

- Internal Land Simulation Option: With this option, the land module computes the hydrograph and pollutograph for each catchment using algorithms adapted from the SWMM (version 5.0.009) land surface compartment and sediment algorithms adapted from the HSPF model, and generate flow and water quality time series for each and every catchment. The time series are then used to drive the routing and BMP simulation.
- External Land Simulation Option: With this option, externally generated time series are used to represent hydrology and water quality at the landscape level. The runoff boundary condition is represented using text files containing a unit-area hydrograph and pollutograph(s) for each land use type, generated from a precalibrated external watershed model such as HSPF (Hydrologic Simulation Program FORTRAN), LSPC (Loading Simulation Program in C++), or any continuous-simulation model capable of generating hourly (or subhourly) time series. SWMM can also be applied and linked to *SUSTAIN* using the external land simulation option, which saves the overhead of having to generate runoff and pollutant loads as part of the optimization simulation.

SUSTAIN BMP Models

BMP Type Classifications

The BMP module is designed to provide a process-based simulation of flow and pollutant transport routing for a wide range of structural BMPs. It is also modular so that new best management practices (BMPs) and alternative solution techniques can be added over time as needed. Depending on the simulation processes involved, BMPs are classified into five types: Type A, Type B, Type C, Type D, and Type E. Table 1 lists the BMP type classification, example practices, and relevant simulation processes. The simulation processes are discussed in details in BMP Simulation Components section of this report.

Table 1. BMP type classifications

BMP type	Example practices	Simulation processes
Type A	Bioretention	• Stage-outflow storage routing using weir or orifice equations • Pump-curve outflow • Green-Ampt/Holtan/Horton • Complete mixed/continuously stirred tank reactor (CSTR) in series • 1st order decay • Kadlec and Knight method
	Cistern	
	Constructed Wetland	
	Green Roof	
	Infiltration Basin	
	Infiltration Trench	
	Porous Pavement	
	Rain Barrel	
	Sand Filter (non-surface)	
	Sand Filter (surface)	
	Wet Pond	
	Dry Pond	
	Regulator	
Type B	Grassed Swale	• Kinematic flow routing by solving the coupled continuity equation and Manning's equation • Green-Ampt/Holtan/Horton • Completely mixed • 1st order decay • Kadlec and Knight method
Type C	Conduit	Conduit simulation involves kinematic wave flow routing and pollutant routing as implemented in SWMM and sediment transport as implemented in HSPF. See detailed descriptions in *SUSTAIN* report (USEPA 2009)
Type D	Filter strip	• Nonlinear reservoir flow routing • Green-Ampt/Holtan/Horton • 1st order decay • Kadlec and Knight method
Type E	Impervious Area Disconnection (Area BMP)	• Nonlinear reservoir flow routing • Steady state infiltration • 1st order decay • Kadlec and Knight method

BMP Simulation Components

The BMP module simulates the following hydrologic processes to reduce land runoff volume and attenuate peak flows: evaporation of standing surface water, infiltration of ponded water into the soil media, deep percolation of infiltrated water into groundwater, and outflow through weir or orifice control structures. Figure 2 shows a schematic of the BMP simulation processes.

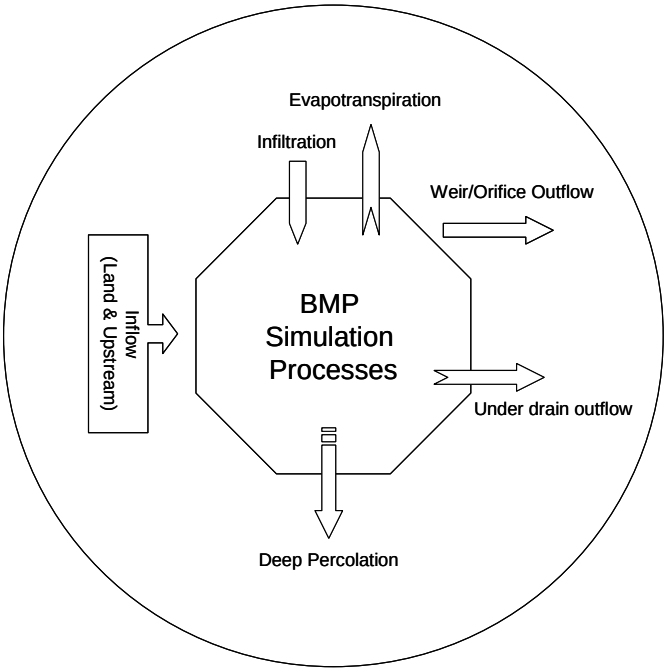


Figure 2. A schematic showing the BMP simulation processes modeled in *SUSTAIN*.

Table 2 provides an overview of the required inputs, the methods used to manage and process the inputs, and the resulting outputs of the BMP module. To help manage model scale, *SUSTAIN* also allows the user to apply an aggregate BMP, which defines a BMP treatment train template to a larger portion of land use within a subcatchment. For example, if a rain barrel is in series with a rain garden at the lot level, and the user wishes to apply the same treatment train to multiple lots, in lieue of specifying every single rain barrel and rain garden, the aggregate BMP approach allows the user to simulate one template that can be replicated as many times as necessary in the subwatershed. The aggregate BMP approach recognizes that not all the drainage area can be treated. While some of the land can be defined as tributary to individual BMP components in the train, any untreatable or untreated areas will be routed directly to the watershed outlet. Figure 3 shows an example of an aggregate BMP treatment train and contributing area distribution, with untreated land area routed directly to the outlet. Some resolution is sacrificed using that approach because a fixed land use distribution and average size for the lot must be assumed; however, this is offset by simulation efficiencies gained. Alternatively, the user can always manually delineate the subareas (BMP drainage areas) within a subcatchment and assign individual BMP to each drainage area if that level of resolution is needed.

Table 2. Summary of inputs, methods, and outputs in the BMP module

BMP Module
Inputs – Define BMP dimensions – Define substrate (soil and underdrain media) properties – Define sediment settling and transport parameters – Define pollutant removal and routing parameters – Define cost for each functional component of a BMP – Sub-hourly inflow time series – Sub-hourly sediment (sand, silt, and clay) concentration time series – Sub-hourly pollutant concentration time series
Methods – Evapotranspiration (ET) is calculated (user-selected constant, monthly, or daily values; derived from daily temperature using Hamon method) – Infiltration is computed using the Green-Ampt or Holtan or Horton methods – Deep percolation is calculated according to user-specified background infiltration rate – Surface outflow is computed using weir or orifice equations – Underdrain outflow is computed using orifice equation – Sediment (sand, silt, and clay) settling and routing is computed using the process-based algorithms adopted from the HSPF model – Pollutant removal is calculated using 1st order decay or Kadlec and Knight method – Pollutant routing is computed using completely mixed or CSTR in series method
Outputs – Sub-hourly outflow time series – Sub-hourly sediment (sand, silt, and clay) concentration time series – Sub-hourly pollutant concentration time series

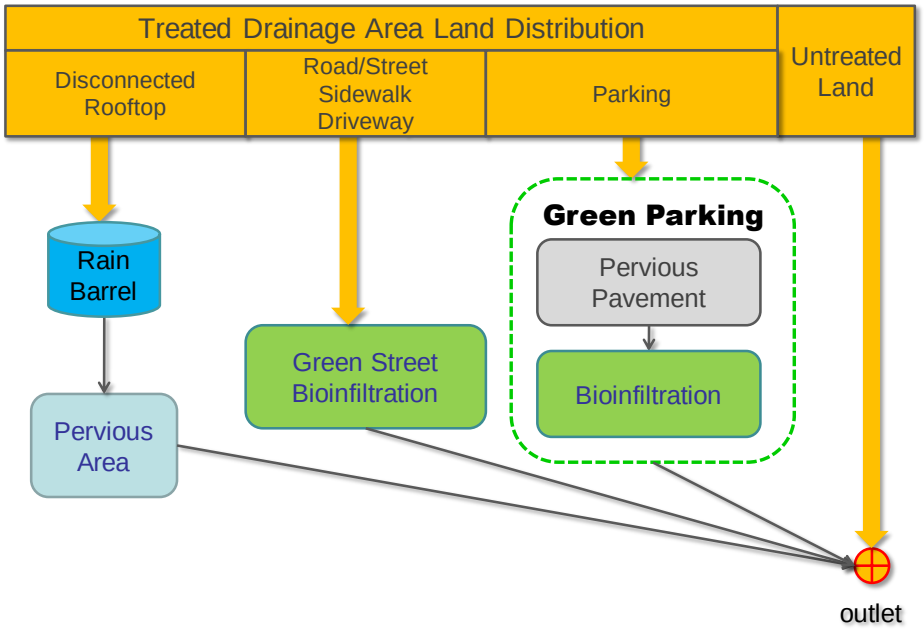


Figure 3. Example of an aggregate BMP treatment train and contributing area distribution.

Table 3 provides a summary of the key BMP simulation processes included in *SUSTAIN*.

Table 3. Available optional methods for BMP simulation processes

Processes	**Options**
Flow Routing	Stage-outflow storage routing using weir or orifice equations Pump-curve outflow For swale: kinematic routing by solving the coupled continuity equation and Manning's equation For filter strip and area BMP: Nonlinear reservoir flow routing
Infiltration	Green-Ampt method Holtan-Lopez equation Horton method
Evapotranspiration	Constant potential evapotranspiration (PET) rate or monthly average value, or daily values Calculate PET using Hamon's method
Pollutant Routing	Completely mixed, single CSTR CSTRs in series
Pollutant Removal	1st order decay *k'-C** method

BMPs in *SUSTAIN* are simulated using a combination of fundamental algorithms to represent the processes of flow routing, infiltration, ET, underdrain infiltration, and pollutant routing and removal. The fundamental algorithms associated with each method are described in more detail below.

Flow Routing

Storage Routing Method

Water balance storage routing is a commonly used method for flow routing in ponds and impoundments.

$$\Delta V/\Delta t = I - O \quad \textbf{(1)}$$

where

ΔV = change in storage (volume),
Δt = time interval (time),
I = inflow (volume per unit time), and
O = outflow (volume per unit time).

Stage-outflow relationships are widely used for flow routing through an orifice or over a weir as shown in Figure 4.

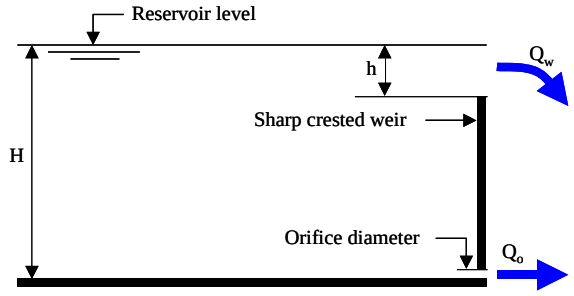


Figure 4. A weir and orifice stage-outflow representaion in *SUSTAIN*.

Weir Outflow

Three commonly used weir types (i.e., sharp-crested rectangular weir, sharp-crested triangular weir, and broad-crested rectangular weir) are supported in *SUSTAIN*.

The equation for the rectangular, sharp-crested weir overflow is (Linsley et al. 1992)

$$Q_w = C_w L_w h^{3/2} \quad \textbf{(2)}$$

where

Q_w = outflow over sharp-crested weir (ft^3/s),
C_w = coefficient of discharge,
L_w = length of weir crest (ft), and
h = depth of the water above weir crest (ft).

Values of C_w (English units) for sharp-crested rectangular weirs are given in Table 4.

Table 4. Coefficient *Cw* (in English units) for rectangular sharp-crested weirs

	Head *h* on Weir, ft						
H_d/h	**0.2**	**0.4**	**0.6**	**0.8**	**1.0**	**2.0**	**5.0**
0.5	4.18	4.13	4.12	4.11	4.11	4.10	4.10
1.0	3.75	3.71	3.69	3.68	3.68	3.67	3.67
2.0	3.53	3.49	3.48	3.47	3.46	3.46	3.45
10	3.36	3.32	3.30	3.30	3.29	3.29	3.28
∞	3.32	3.28	3.26	3.26	3.25	3.25	3.24

Source: Linsley et al. 1992
H_d = Height of the weir

The equation for the triangular (V-notch) sharp-crested weir overflow is (Linsley et al. 1992)

$$Q_w = C_w'' \frac{8}{15}\sqrt{2g}\, h^{5/2} \tan\left(\frac{\theta}{2}\right) = 4.28 C_w'' h^{5/2} \tan\left(\frac{\theta}{2}\right) \quad \textbf{(3)}$$

where

Q_w = outflow over sharp-crested weir (ft^3/s),
C_w'' = coefficient of discharge, default value is 0.58 for English units,
h = depth of the water above weir crest (ft),
θ = vertex angle of the V-notch, and
g = acceleration of gravity (32.2 ft/s^2).

True broad-crested weir flow occurs when the upstream head above the crest is between about 1/20 and 1/2 the crest length in the direction of flow (USBR 2001). For broad-crested weirs, it is recommended that weir coefficient C_w be determined by measuring the flow at various flow rates (Linsley et al. 1992). The value of the weir coefficient varies with h/H_d.

$$C_w = \frac{0.65}{\left(1+\frac{h}{H_d}\right)^{1/2}} \frac{2}{3}\sqrt{2g} \quad \textbf{(4)}$$

where

h = depth of the water above the weir crest (ft),
H_d = height of the weir (ft), and

g = acceleration of gravity (32.2 ft/s^2).

Orifice outflow

The equation for the orifice flow is

$$Q_o = C_o A_o \sqrt{2gH} \quad \textbf{(5)}$$

where

Q_o = outflow through orifice (ft^3/s),
C_o = orifice coefficient of discharge,
A_o = orifice cross sectional area (ft^2),
g = acceleration due to gravity (ft/s^2), and
H = depth of the water level above the orifice (ft).

Nonlinear Reservoir Overland Flow Routing Method

A nonlinear reservoir routing algorithm is applied to route the surface runoff from the impervious area to the Area BMP (i.e., pervious area), also the overland flow through buffer strips. Surface runoff, Q, occurs only when the surface water depth, d, exceeds the maximum surface storage depth, d_p, in which case, the outflow is given by Manning's equation:

$$Q = W \frac{1.49}{n} \left(d - d_p\right)^{5/3} S^{1/2} \quad \textbf{(6)}$$

where

Q = outflow rate (cfs),
W = pervious area width (ft),
n = Manning's roughness coefficient,
d = water depth (ft),
d_p = depth of surface storage (ft), and
S = pervious area slope (ft/ft).

Pump Curve

Certain management practices require an external pump to convey flow out of the BMP. Pump curves define the numeric relationship between BMP water depth and pump flow rate, similar to the Type 4 pump curve available in SWMM (Rossman 2005). Figure 5 presents a conceptual illustration of a pump implemented in a storage tank.

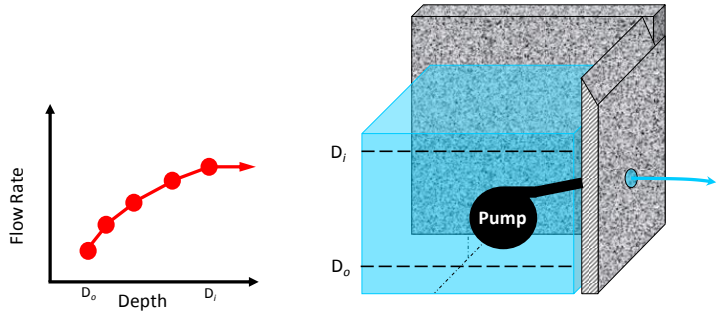


Figure 5. Conceptual illustration of the BMP pump curve.

The curve is represented as a table of paired water depth and flow rate values. The water depths represent the pump's operating bounds, where D_o is the depth of the pump's minimum operating

capacity and D_i is the depth of the pump's maximum operating capacity after which flow rate becomes constant. The pump can be implemented in a BMP that also has orifice, weir, or underdrain; however, the pump demand will take priority over the outlets.

Infiltration/Filtration

SUSTAIN supports three options for simulating infiltration in BMPs: (1) Holtan-Lopez equation, (2) Green-Ampt equation, and the (3) Horton equation.

Holtan-Lopez Empirical Model

The Holtan-Lopez empirical model computes the infiltration rate as a function of the actual available soil water storage, S_a, of the surface soil layer, as shown below (Maidment 1993):

$$f = GRI \times A \times S_a^{1.4} + f_c \quad \textbf{(7)}$$

where

f = infiltration rate (in./hr),
GRI = growth index of vegetation in percent maturity, varying from 0.1 to 1.0,
A = infiltration capacity (an index representing surface-connected porosity and density of plant roots),
S_a = available storage in the surface layer (in.), and
f_c = constant final infiltration rate (in./hr).

In Equation (7), A is the vegetative parameter that characterizes surface-connected porosity and the density of plant roots, which affect infiltration (a value of 0.8 is a typical number for sod or vegetation that would be found in a BMP). f_c is the final constant infiltration rate (in./hr), which is a function of the hydrologic soil group. The value of f_c ranges from 0.3 in./hr for group-A soils to between 0.0 and 0.05 in./hr for group-D soils (Maidment 1993). In a continuous calculation, the available soil storage (S_a) and infiltration rate (f) are computed at each simulation time step. Available soil storage is updated each time increment and the infiltration is calculated. The available storage in the surface layer is recovered through evapotrapsiration between storm events.

This method was developed using the premise that soil moisture storage, surface-connected porosity, and the effect of root density of the control soil layer are the dominant factors influencing the infiltration process.

A difficulty with using this method is estimating the control soil layer depth. For simulating the infiltration process, it is assumed that the soil column depth is the control depth because BMP devices normally have a confined soil/substrate layer.

Green-Ampt Infiltration Equation

The Green-Ampt infiltration method assumes that a sharp wetting front exists in the soil column that separates the unwetted zone of soil with some initial moisture content below and the wetted zone of soil above (Rossman 2005). The infiltration rate is calculated as a function of soil moisture, saturated hydraulic conductivity, and average wetting front suction head, and is based on Darcy's law and the principle of mass conservation (Huber and Dickinson 1988).

If $I \le K_s$, then $f = I$.

If $I > K_s$, then $f = I$, until $F = F_s = \dfrac{(\theta_s - \theta_i) \times \psi_f}{1 - I/K_s}$.

After surface saturation,

$$f = \frac{dF}{dt} = K_s\left[1 + \frac{(\theta_s - \theta_i)\times\psi_f}{F}\right] \quad \textbf{(8)}$$

For $I > K_s$, and $f = I$ for $I \le K_s$

where

I = inflow rate (in./hr),
F = amount of infiltration (in.),
F_s = amount of infiltration up to surface saturation (in.),
F = infiltration rate (in./hr),
K_s = saturated hydraulic conductivity (in./hr),
θ_s = saturated moisture content,
θ_i = initial moisture content, and
ψ_f = average wetting front suction head (in. of water).

This differential equation is solved iteratively to determine f at each time step by using Newton-Raphson method. The infiltration volume during the time interval is equal to the inflow volume if the surface does not saturate. If saturation occurs during the time interval, the infiltration volumes over each stage of the process within the time steps are calculated and summed. When there is no inflow, any water ponded on the surface is allowed to infiltrate and added to the cumulative infiltration volume. The soil moisture content is changed through soil evapotranspiration and percolation losses between storm events. In using the Green-Ampt method, a complication occurs when the inflow rate starts at a value above, drops below, and then rises above K_s again during the infiltration computation. In such a case, the moisture content must be redistributed as the assumption of saturation from the surface down to the wetting front does not hold. When performing BMP infiltration simulation, the impact of the underdrain layer, the impermeable bottom layer, or both, on the infiltration process must be considered in the simulation. Because the Green-Ampt method can be applied to a layered soil column, the underdrain layer can be represented as a separate layer under the soil column. In cases where an impermeable layer is present at the bottom of the soil column, the infiltration rate ceases when the soil storage capacity is reached. A drawback of the Green-Ampt method is that it does not include a parameter to explicitly reflect the effect of the vegetation root zone on the infiltration rate.

Horton Infiltration Method

The Horton infiltration method is implemented in *SUSTAIN* using the Storm Water Management Model (SWMM) formulation (Rossman 2005). The Horton infiltration method is an empirically based model parameterized by specifying an initial (maximum) infiltration rate and a final, saturated infiltration rate. The model assumes that infiltration begins at a constant, maximum rate that decreases exponentially over time. The shape of the curve as the infiltration rate changes from initial to final is controlled by a decay rate specific to the type of soil (USEPA 1998). The equation follows:

$$f_t = f_c + (f_o - f_c)e^{-kt} \quad \textbf{(9)}$$

where f_t is the infiltration rate at time t, f_o is the maximum infiltration rate, f_c is the saturated infiltration rate, and k is the decay constant.

The present time (t) on the infiltration curve between the storm events is regenerated using a regeneration coefficient of infiltration rate based on the user-specified number of drying days along an exponential drying curve.

Evapotranspiration

PET time series can be estimated using the U.S. Weather Bureau Class A pan records with adjustment for plant influences. Several methods are also available to estimate PET. The Penman-Monteith method is a comprehensive energy-balance approach (Maidment 1993) that requires time series values for solar radiation, air temperature, relative humidity, and wind speed. The Priestley-Taylor method (Maidment 1993) requires solar radiation, air temperature, and relative humidity. The Hargreaves method (Maidment 1993) requires only air temperature. This method estimates reference crops' evapotranspiration on a monthly or larger time interval and can be used when climatological data are limited. The Hamon method (1961) also requires only air temperature but estimates daily PET.

SUSTAIN provides three options to estimate PET: (1) apply 12 user-supplied monthly PET rates (2) calculate PET from a user-supplied pan evaporation time series input and a monthly pan coefficient that is a function of BMP vegetation, or (3) internally calculate PET rates for each hour using Hamon's method (1961).

As previously noted, Hamon's (1961) method generates daily PET using air temperature, a monthly variable coefficient, and absolute humidity (computed from air temperature). The PET value is computed as follows:

$$PET = C_{TS} \times D_{hr}{}^{2} \times \rho_{ws} \quad \textbf{(10)}$$

where

PET = daily PET (in.),

C_{TS} = monthly variable coefficient, and

D_{hr} = possible hours of sunshine computed as a function of latitude and time of year.

$$\rho_{ws} = \frac{216.7 \times p_{ws}}{T_{av} + 273.3} \quad \textbf{(11)}$$

where

ρ_{ws} = saturated water vapor density (absolute humidity) at daily mean air temperature (g/cm^3) and

T_{av} = mean daily air temperature (˚C).

$$p_{ws} = 6.108 \times exp\left[\frac{17.26939 \times T_{av}}{T_{av} + 273.3}\right] \quad \textbf{(12)}$$

where

p_{ws} = saturated vapor pressure at the air temperature and

T_{av} = mean daily air temperature (˚C).

Hamon (1961) suggests a constant value of 0.0055 for C_{TS}. However, monthly values can be specified to avoid underestimating PET in some areas, especially for the winter months.

Calculate Actual Evapotranspiration

Once PET is determined, the actual evapotranspiration (ET) is calculated as a function of PET and soil moisture storages. While PET represents the maximum possible achievable ET on the basis of atmospheric conditions alone, actual ET is determined using an accounting of the status of the various components of the hydrologic budget. The actual ET is equal to PET when the soil moisture is greater than or equal to the moisture at the field capacity. There is no actual ET if the moisture content is less than or equal to the moisture at the wilting point.

Evapotranspiration (ET) Multiplier

An ET multiplier was added allowing a unique multiplier value to be set for each BMP instance in a *SUSTAIN* model. The multiplier is applied to the global ET rate (e.g., constant monthly, time series, and so on) to account for unique ET conditions that are BMP specific. For example, a multiplier greater than one can be used to parameterize an individual BMP with more abundant or efficient vegetation that tends to enhance ET above and beyond direct pan evaporation.

Underdrain Method

Underdrain Outflow

The underdrain outflow in a BMP is modeled using a simple water balance concept. The available underdrain storage is represented as the total of void spaces beneath the upper soil layer. Inflow into underdrain storage is limited by the final saturated infiltration rate of the upper soil layer. Because the primary function of the underdrain is to provide additional water storage and to delay outflow, the outflow pipe draining the underdrain layer is placed at the interface between the upper soil layer and the underdrain layer, creating that supplemental storage compartment in the underdrain layer. Figure 6 illustrates the function of the underdrain together with other substrate model components.

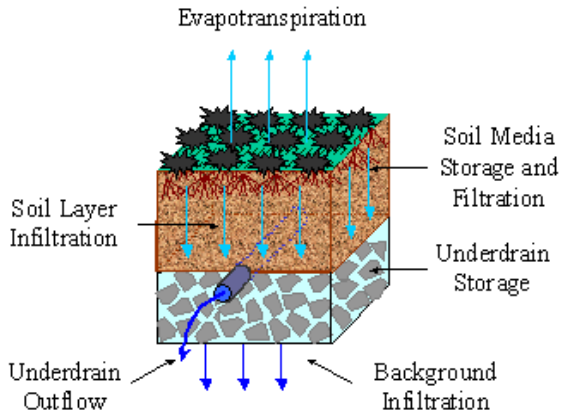


Figure 6. Processes considered in an underdrain structure.

Outflow from the underdrain layer is assumed to be unrestricted; therefore, no pipe outflow is required. Underdrain outflow is part of the modeled BMP effluent and occurs when all available underdrain storage is used up, when the water level meets or exceeds the underdrain level, or when both occur. So underdrain outflow equals the storage zone inflow that exceeds the available underdrain capacity minus the background infiltration—it can never be greater than the saturated layer infiltration rate minus the background infiltration rate. Each infiltration management practice can be modeled with or without underdrain outflow. If underdrain outflow is enabled, the user must specify the thickness of the underdrain storage layer, the media void fraction, and the background infiltration rate (Figure 6). Water and pollutants are removed from the system entirely through background infiltration.

Underdrain Filtration of Pollutants

If underdrain is specified in the *soil properties* of a BMP, an additional reduction in pollutant load leaving the underdrain can be applied. This reduction is applied using the *underdrain percent removal*, which is a user-supplied parameter. This is the only place in *SUSTAIN* where a percent removal is applied. Because the underdrain outflow rate is relatively constant, it is reasonable to apply a percent reduction to represent the impact of soil media filtration. All other pollutant reductions are estimated using a first-order decay rate, which is a loss computed as a function of the retention time of water in the BMP, as discussed below.

Pollutant Routing and Removal Methods

The methods of pollutant routing to achieve pollutant reduction are described in this subsection for a completely mixed system and a multiple impoundments in series. The flow through a plug flow reactor (PFR), as a series of infinitely thin coherent *plugs*, each with a uniform composition, is assumed to be perfectly mixed in the lateral direction, but not in the longitudinal direction (direction of flow). Each plug of differential volume is considered as a separate entity, with an infinitesimally small volume that requires a very small time steps (in seconds) during calculation. Because *SUSTAIN* uses time steps rainging between one minute and one hour to simulate flow and pollutant routing, the current version does not support the plug flow option. However, it can be seen that an infinite number of small continuously stirred tank reactors (CSTRs) operating in series mimics the behavior of a PFR as compared to a single completely mixed segment.

First-Order Decay with Complete Mixing

This method is commonly used and suitable for small ponds where complete mixing is a reasonable simplifying assumption. The numeric expression is as follows:

$$\frac{d(VC)}{dt} = I(t)C_I(t) - O(t)C(t) - KC(t)V(t) \qquad \textbf{(13)}$$

where

V = reservoir volume (ft^3),
C_I = influent pollutant concentration (mg/L),
C = effluent and reservoir pollutant concentration (mg/L),
I = inflow rate (ft^3/s),
O = outflow rate (ft^3/s),
t = time (sec), and
K = decay coefficient (1/s).

Continuously Stirred Tank Reactors in Series and Kadlec and Knight's Model

CSTRs in series are used to represent a hydraulic condition intermediate between a completely mixed and plug flow (Wong et al. 2001, 2002). That method is applied for simulating first-order pollutant removal processes (e.g., settling, decay) that occur in ponds, wetlands, and other similar BMPs. It is not as relevant for soil infiltration systems, such as rain gardens or green roofs because water does not pond for long periods of time. The calculation is a two-step process that begins by estimating the number of reactors in series to be selected to represent the shape of the BMP, and then applying first order kinetics with a nonreactive background concentration (Kadlec and Knight 1996).

Step 1: Estimate N, the number of CSTRs in series.

N, the number of CSTRs in series, can be approximated on the basis of BMP shape (Persson et al. 1999; Wong et al. 2001, 2002). Values of *N* for the various pond shapes, shown in Figure 7, are presented in Table 5. Highest *N* values are for ponds with a distributed inflow (pond E), baffles (pond G), and very elongated flow or high length to width ratio (pond J).

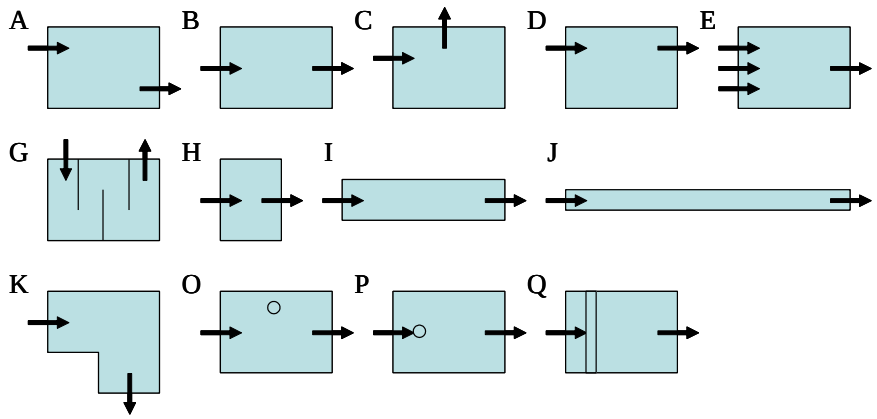


Figure 7. Conceptual pond shapes simulated by Persson et al. (1999).

Table 5. Quality ratings of conceptual pond shapes simulated by Persson et al. (1999)

Pond	N ≈ 1 / (1-λ)	Qualitative rating
J	10.0	Good
G	4.2	
E	4.1	
P	2.6	Satisfactory
Q	2.5	
I	1.7	Poor
K	1.6	
A	1.4	
B	1.4	
O	1.3	
D	1.2	
H	1.1	
C	1.1	

Step 2: Apply first-order decay to each CSTR.

After selecting the number of reactors, pollutants are modeled for each tank at each time step using the first-order kinetic model, described in Equation (14).

$$(C_{out} - C^*)/(C_{in} - C^*) = e^{-k'/q} \quad \textbf{(14)}$$

where

C^* = background concentration (mg/L),
C_{in} = input concentration (mg/L),
C_{out} = output concentration (mg/L),
q = hydraulic loading or overflow rate (m/yr),
$k' = k \cdot h$ = rate constant (m/yr),
k = first order decay rate (1/yr), and
h = pond depth (m).

This equation is computed separately for each time step at each CSTR. The main difference between this equation and ordinary first-order decay modeling for a CSTR is the inclusion of C^*, the background concentration, below which the effluent cannot be reduced. Another advantage of this method is that

using an *areal* rate constant (units of depth/time) instead of a *volumetric* one (units of inverse time) helps avoid having to specify an average depth or the volume for odd natural configurations. Instead, only the pond surface area is required to compute the hydraulic loading rate q.

Wong et al. (2002) recommend some k' and C^* values as shown in Table 6, on the basis of limited model calibrations for total suspended solids (TSS), total phosphorus (TP), and total nitrogen (TN) in urban areas near Melbourne. Those default values should be used with caution because conditions vary by location. However, they could be used as a starting point for baseline model calibration where local values are not available.

Localized calibration can be performed to customize the simulation technique for specific areas. The C^* and k' values can be determined or calibrated using monitoring data (particle size distribution in particular) and treatment measure design specifications.

Table 6. Recommended k' and C^* values

Treatment measures	**k' (m/yr)**			**C^* (mg/L)**		
	TSS	TP	TN	TSS	TP	TN
Sedimentation Basins	15,000	12,000	1,000	30	0.18	1.7
Ponds	1,000	500	50	12	0.13	1.3
Vegetated Swales	15,000	12,000	1,000	30	0.18	1.7
Wetlands	5,000	2,800	500	6	0.09	1.3

Representation of Area BMP – Impervious Disconnection

The Area BMP is a pervious land segment over which a portion of impervious runoff, from disconnected impervious areas like rooftops, is routed. The BMP simulation occurs only when there is no runoff from the BMP area. Otherwise, the total inflow to the BMP is bypassed. The Area BMP simulation is an approximation to the reality where the runoff from the disconnected impervious area is routed to and simulated on the pervious area. The runoff from the disconnected impervious area is captured by the Area BMP through the infiltration (under saturated soil condition) and the surface storage. The runoff from the BMP area (i.e., pervious area) is not simulated by the Area BMP and is always bypassed. Figure 8 shows the conceptual flow diagram of Area BMP simulation.

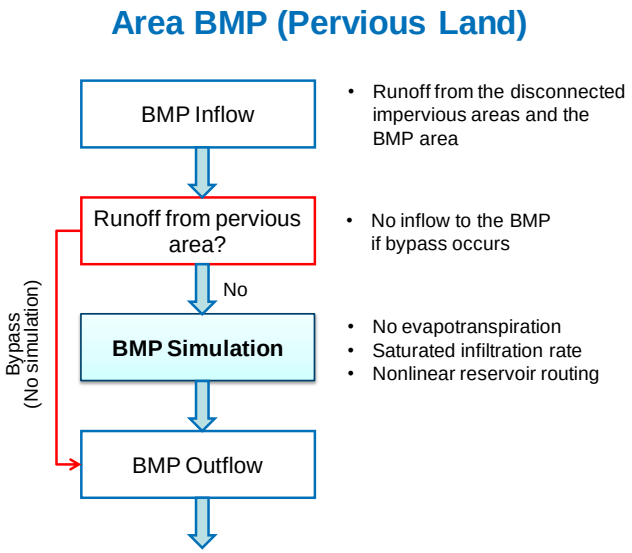


Figure 8. Conceptual flow diagram of Area BMP simulation.

Sediment Transport Simulation

This section describes the transport, deposition, and scour of inorganic sediment through BMPs using the HSPF algorithms (Bicknell et al. 2001). These routines are most applicable for BMP types with significant ponded water depth (e.g., ponds, swales, conduits, and the like). Figure 9 shows the principal state variables and fluxes involved in the sediment transport processes.

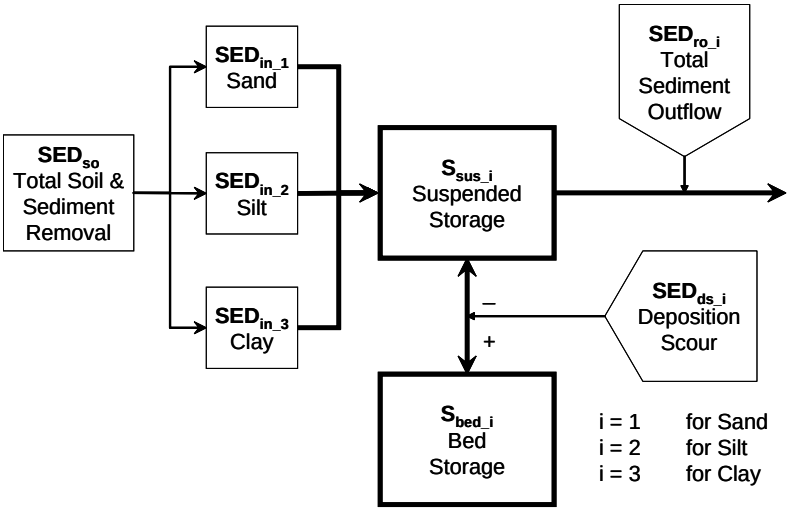


Figure 9. Schematic of sediment transport, deposition, and scour in conduits.

Both the migration characteristics and the adsorptive capacities of sediment vary significantly with particle size. To facilitate analyses to account for the effects of particle sizes, *SUSTAIN* divides the inorganic sediment load into three components (sand, silt, and clay), each with its own properties. Sand has a particle size ranging from 0.05 millimeter (mm) to 2.0 mm in diameter, silt from 0.002 mm to 0.05 mm in diameter, and clay smaller than 0.002 mm. The user specifies the fraction of each component by land use in the surface runoff and the model applies those fractions to distribute the total sediment load into sand, silt, and clay portions from each land use and routes them independently through the BMP network.

The system assumes that scour or deposition of inorganic sediment does not affect the hydraulic properties of the water column (i.e. sediment movement does not change the shape of the channel). Furthermore, it is assumed that sand, silt, and clay deposit independently in the water column bed so that the deposition or scour of one material is not linked to the changes of others. Longitudinal movement of bed sediments by flow shear stress is not modeled, although sediment can be resuspended from one segment and deposited in another one downstream.

First, the volume occupied by each component of bed sediment is calculated as shown in Equation (15).

$$V_{bed\_i} = \frac{S_{bed\_i}}{\rho_{\_i}} \quad \textbf{(15)}$$

where

V_{bed_i} = volume occupied by component *i* of bed sediment (ft^3),
S_{bed_i} = bed storage of component *i* of sediment (lb), and
$\rho_{\_i}$ = particle density of component *i* (lb/ft^3).

The volumes of the three components of bed sediment are summed, and the total bed volume is adjusted to account for voids in the sediment (i.e., the porosity):

$$V_b = \frac{\sum_{i=1}^{i=3} V_{bed\_i}}{1-\eta} \quad \textbf{(16)}$$

where

V_b = volume of bed (ft^3),
V_{bed_i} = volume of sediment contained in the bed (sand, silt, and clay) (ft^3), and
η = porosity of bed sediment (ratio of pore volume to total volume).

Finally, the depth of bed sediment is calculated as

$$d_b = \frac{V_b}{L_r \times W_b} \quad \textbf{(17)}$$

where

d_b = depth of bed (ft),
V_b = volume of bed (ft^3),
L_r = length of water column (ft), and
W_b = effective width of bed (ft).

Cohesive sediments

Two steps are used to model the deposition, scour, and transport processes of cohesive sediments (silt and clay). The first step computes the advective transport, and the second step calculates the amount of deposition or scouring on the basis of the bed shear stress.

Advective Transport of Constituent

This step computes the concentration of material in a water column and the quantities of material that leave the water column due to longitudinal advection. Two assumptions are made in the solution technique for normal advection: (1) each constituent is uniformly dispersed throughout the waters of the water column and (2) the constituent is completely entrained by the flow—that is, the material moves at the same horizontal velocity as the water.

The equation of continuity can be written as

$$SED_{in} - SED_{ro} = (C \times V) - (C_s \times V_s) \quad \textbf{(18)}$$

where

SED_{in} = total inflow of sediment over the interval (lb),
SED_{ro} = total outflow of sediment over the interval (lb),
C_s = sediment concentration at the start of the interval (lb/ft^3),
C = sediment concentration at the end of the interval (lb/ft^3),
V_s = volume of water stored at the start of the interval (ft^3), and
V = volume of water stored at the end of the interval (ft^3).

The other basic equation states that the total outflow of material over the time interval is a weighted mean of two estimates; one based on conditions at the start of the interval, the other on ending conditions:

$$SED_{ro} = (C_s \times Q_s \times js) + (C \times Q \times cojs) \quad \textbf{(19)}$$

where

Q_s = outflow rate at the start of the interval (ft^3/time interval),
Q = outflow rate at the end of the interval (ft^3/time interval),
js = weighting factor, and
cojs = *1 – js*.

By combining equations (18 and (19), we can solve for the concentration *C*:

$$C = \frac{SED_{in} + C_s \times (V_s - Q_s \times js)}{V + Q \times cojs} \quad \textbf{(20)}$$

The total amount of material leaving the water column during the interval is calculated using Equation (19). If the water column goes dry during the interval, the total amount of material leaving the water column is the sum of the material coming in and the material leaving according to the concentration at the start of the interval:

$$SED_{ro} = SED_{in} + (C_s \times Q_s \times js) \quad \textbf{(21)}$$

Deposition and Scouring

Exchange of cohesive sediments with the bed is dependent on the shear stress exerted on the bed surface. When the shear stress (τ) in the water column is less than the user-supplied, critical, shear stress for deposition (τ_{cd}), sediment deposition occurs. On the other hand, when the shear stress is greater than the user-supplied, critical, shear stress for scour (τ_{cs}), scouring of cohesive bed sediments takes place. The rate of deposition for a fraction of cohesive sediment is based on a simplification of Krone's equation in the following form:

$$D = \omega \times C \times \left(1 - \frac{\tau}{\tau_{cd}}\right) \quad \textbf{(22)}$$

where

D = rate at which sediment settles out of suspension (lb/ft^2 interval),
ω = settling velocity for cohesive sediment (ft/interval),
C = concentration of suspended sediment (lb/ft^3),
τ = shear stress (lb/ft^2), and
τ_{cd} = critical shear stress for deposition (lb/ft^2).

The rate of change of suspended sediment concentration in the water column due to deposition can be expressed as follows:

$$\frac{dC}{dt} = -\frac{D}{d_{av}} \quad \textbf{(23)}$$

where

d_{av} = average depth of water in the water column (ft).

By substituting the expression for deposition rate (D) from Equation (22), and integrating and rearranging Equation (23), a solution can be obtained for the concentration of suspended sediment lost to deposition during a simulation interval (C_{dep}):

$$C_{dep} = C \times \left[1 - exp\left\{ \left(-\frac{\omega}{d_{av}} \right) \times \left(1 - \frac{\tau}{\tau_{cd}} \right) \right\} \right] \quad \textbf{(24)}$$

where

C = concentration of suspended sediment at the start of interval (lb/ft^3),
ω = settling velocity for sediment fraction (ft/interval),
d_{av} = average depth of water in water column (ft),
τ = shear stress (lb/ft^2), and
τ_{cd} = critical shear stress for deposition (lb/ft^2).

The user must supply values for settling velocity (ω) and critical shear stress for deposition (τ_{cd}) for silt and clay fractions in cohesive sediment.

The amount of sediment in suspension (S_{sus}) is updated by subtracting the amount settled. Likewise, the amount of sediment in bed (S_{bed}) is updated by adding the amount settled on it.

The rate of resuspension, or scour, of cohesive sediments from the bed is derived from a modified form of Partheniades' (1962) equation:

$$S = \mu \times \left(\frac{\tau}{\tau_{cs}} - 1 \right) \quad \textbf{(25)}$$

where

S = rate at which sediment is scoured from the bed (lb/ft^2 interval),
μ = erodibility coefficient for the sediment fraction (lb/ft^2 interval), and
τ_{cs} = critical shear stress for scour (lb/ft^2).

The rate of change of suspended sediment fraction concentration in the water column due to scour can be expressed as follows:

$$\frac{dC}{dt} = -\frac{S}{d_{av}} \quad \textbf{(26)}$$

By substituting the expression for scour rate *(S)* from Equation (25) and integrating and rearranging Equation (26), a solution can be obtained for the concentration of suspended sediment added to suspension by scour during a simulation interval (C_{scr}):

$$C_{scr} = \frac{\mu}{d_{av}} \times \left[\frac{\tau}{\tau_{cs}} - 1 \right] \quad \textbf{(27)}$$

where

μ = erodibility coefficient (lb/ft^2 interval), and
d_{av} = average depth of water (ft).

The user is required to supply values for the erodibility coefficient (μ) and critical shear stress for scour (τ_{cs}) for each fraction of cohesive sediment (silt and clay) that is modeled.

The amount of sediment in suspension (S_{sus}) is updated by adding the scoured mass, as is the amount of sediment in bed (S_{bed}) by subtracting the scoured mass.

If the amount of scoured sediment is greater than the original sediment in the bed, all sediment in the bed will be resuspended and the amount of sediment in the bed is set to zero.

Non-Cohesive Sediment

Erosion and deposition of sand, or non-cohesive sediment, is affected by the amount of sediment that the flow is capable of carrying. If the amount of sand being transported is less than the flow can carry for the hydrodynamic conditions of the water column, sand is scoured from the bed. This occurs until the actual sand transport rate becomes equal to the carrying capacity of the flow or until the available bed sand is all scoured. Conversely, deposition occurs if the sand transport rate exceeds the flow's carrying capacity.

The sand transport capacity for a water column is calculated by using an input power function of the velocity. The potential sand concentration (C_p) is determined by the following conversion:

$$C_p = k \times \upsilon_{av}{}^{j} \quad \textbf{(28)}$$

where

C_p = potential sand concentration (lb/ft^3),
k = coefficient in the sandload suspension equation (input parameter),
j = exponent in sandload suspension equation (input parameter), and
υ_{av} = average velocity (ft/s).

The potential outflow of sand (SED_{pro}) is calculated as follows:

$$SED_{pro} = \left(C_s \times Q_s \times js\right) + \left(C_p \times Q \times cojs\right) \quad \textbf{(29)}$$

where C_s, Q_s, js, Q, and $cojs$ are as previously defined for equations (18) and (19).

The potential scour from, or deposition to, the bed storage is found using the continuity equation:

$$SED_{pds} = \left(V \times C_p\right) - \left(V_s \times C_s\right) + SED_{pro} - SED_{in} \quad \textbf{(30)}$$

where

SED_{pds} = potential scour (+) or deposition (-) (lb),
C_p = potential sand concentration at the end of the interval (lb/ft^3),
C_s = sand concentration at the start of the interval (lb/ft^3),
SED_{pro} = potential outflow of sand over the interval (lb), and
SED_{in} = inflow of sand during the interval (lb).

The potential scour is compared to the amount of available sand for resuspension. If scouring potential is less than the available sands, the demand is satisfied in full and the bed storage is adjusted accordingly. If the potential scour cannot be satisfied by bed storage, all the available bed sand is suspended, and the bed storage is exhausted. The concentration of suspended sand (C) is calculated as follows:

$$C = \frac{SED_{in} + SED_{ds} + C_s \times \left(V_s - Q_s \times js\right)}{V + Q \times cojs} \quad \textbf{(31)}$$

where

C = concentration of sand at end of interval (lb/ft^3),

C_s = concentration of sand at start of interval (lb/ft^3),
SED_{in} = inflow of sand during the interval (lb), and
SED_{ds} = sand scoured from, or deposited to, the bottom (lb).

The total amount of sand leaving the water column during the interval is calculated using equation (31). If water column goes dry during an interval, or if there is no outflow, all the sand in suspension at the beginning of the interval is assumed to settle out, and the bed storage is correspondingly increased.

Sediment Transport Input Parameters

Parametric information required for silt and clay includes particle diameter (ϕ), particle settling velocity in still water (ω), particle density (ρ), critical shear stress for deposition (τ_{cd}), critical shear stress for scour (τ_{cs}), and erodibility coefficient (μ). Parameter values required for sand include median bed sediment diameter (ϕ_{50}) and particle settling velocity (ω).

SUSTAIN Optimization Model

SUSTAIN includes an optimization module to develop cost-effective BMP placement and selection strategies on the basis of a preselected list of potential sites and applicable BMP types and size ranges. The module uses evolutionary optimization techniques to perform the searches for optimal combinations of BMPs that meet the user-defined decision criteria. Table 7 summarizes the required inputs, methods used, and outputs and Figure 10 presents a conceptual overview of the module.

The optimization module works hand-in-hand with the BMP, land, and conveyance modules during the search process iteratively and evolutionarily. The simulation modules evaluate the BMP performance, as defined via evaluation factors, and cost data of a set of chosen BMP options and pass that information to the optimization engine. The optimization engine synthesizes the information, modifies the search path, and generates new solutions that are repeatedly evaluated using the simulation modules. Through this evolutionary search process, the module progressively marches toward indentifying the best or most cost-effective BMP solutions that meet the user's conditions and objectives.

Table 7. Summary of inputs, methods, and outputs in the optimization module

Inputs – Define decision variables (the size ranges of potential BMPs) – Define assessment point(s) and evaluation factor(s) – Define management targets (for the *minimize cost* option) – Define BMP cost functions
Methods – For the *minimize cost* option, optimization search is performed using the Scatter Search technique – For the *generate cost-effectiveness curve* option, optimization search is performed using the NSGA-II technique
Outputs – For the *minimize cost* option, the optimization process outputs optimal solutions that meet the specified treatment targets – For the *cost-effectiveness curve* option, the optimization process outputs the optimal solutions along the cost-effectiveness curve

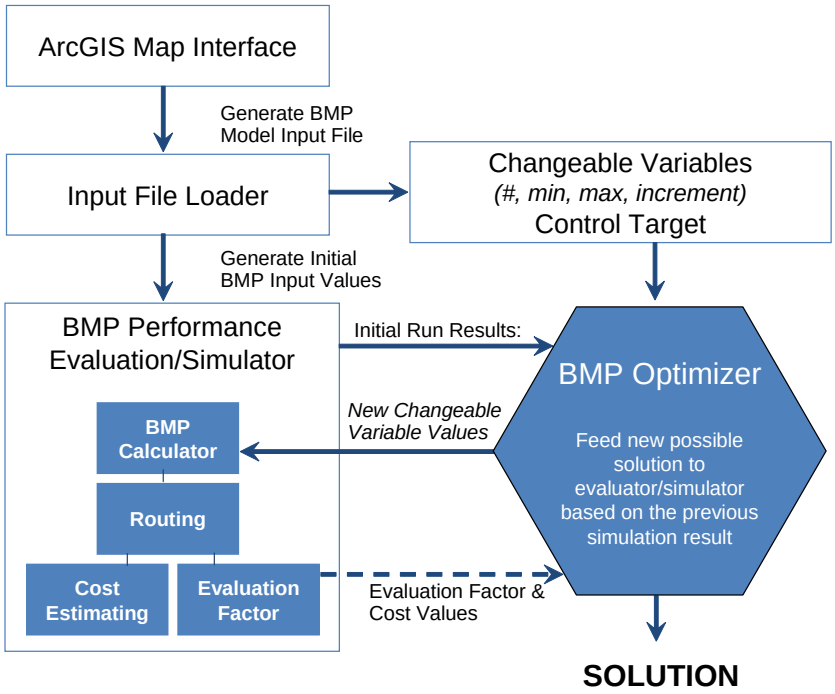


Figure 10. Conceptual overview of the optimization module.

Optimization Setup

The objective of the optimization module is to determine BMP locations, types, and design configurations that minimize the total cost of management while satisfying water quality and quantity constraints. To formulate an optimization problem, *SUSTAIN* requires the user to specify this information: decision variables, BMP cost functions, assessment points and evaluation factors, and management targets.

Decision Variables

Placing BMPs at different spatial levels or locations (or both) affects the overall cost-effectiveness of the stormwater control system (Zhen and Yu 2004). Therefore, BMP location represents one important decision variable for optimization. The possible BMP locations are typically preselected on the basis of multiple factors, including availability of space site characteristics (slope, soil infiltration rates, and water table elevation) and other logistical considerations. Another important decision variable involves BMP configuration. At a given feasible location of a BMP type, the configuration parameters can be treated as decision variables with the specified minimum, maximum, and discrete search interval values. The user can specify the minimum decision variable value as zero, which can be selected if that BMP location is not required to meet the evaluation target. To optimize how much drainage area is to be treated, the user can define a fixed-dimension BMP with a specified design drainage area. In this case, the maximum number of BMPs is the decision variable. The maximum treated drainage area equals the design drainage area times the maximum number of BMPs. The optimizer determines the number of BMP-and-drainage-area units needed to achieve the target—all unteated drainage area is routed to the watershed outlet.

BMP Cost Function

The total cost of a solution is the sum of costs of every BMPs in the project. The cost of each BMP is represented using a generic cost function:

$$Cost\ (\$) = (LinearCost \times Length^{LengthExp} + AreaCost \times Area^{AreaExp} + TotalVolumeCost \times TotalVolume^{TotalVolExp} + MediaVolumeCost \times SoilMediaVolume^{MediaVolExp} + UnderDrainVolumeCost \times UnderDrainVolume^{UDVolExp} + ConstantCost) \times (1+PercentCost/100)$$

where

LinearCost = cost per unit length of the BMP structure ($/ft),
AreaCost = cost per unit area of the BMP structure ($/ft^2),
TotalVolumeCost = cost per unit total volume of the BMP structure ($/ft^3),
MediaVolumeCost = cost per unit volume of the soil media ($/ft^3),
UnderDrainVolumeCost = cost per unit volume of the under drain structure ($/ft^3),
LengthExp = exponent for linear unit,
AreaExp = exponent for area unit,
TotalVolExp = exponent for total volume unit,
MediaVolExp = exponent for soil media volume unit, and
UDVolExp = exponent for underdrain volume unit.
ConstantCost = constant cost ($) (e.g., land cost),
PercentCost = costs expressed as a percentage of all other costs, such as engineering and design cost, project management cost, operational and maintenance costs, and the like (%). This multiplier is applied to the total cost.

Assessment Points and Evaluation Factors

An assessment point is a location where the water quantity or quality or both are evaluated. Any node in the BMP or flow network can be an assessment point. Figure 11 shows an example of assessment points that can be at the watershed outlet, key tributary outlets, or the most downstream node of a stream segment. It is even possible to define a virtual outlet in a network and use it as an assessment point. A virtual outlet is a node where multiple disconnected areas can be routed for evaluation purposes.

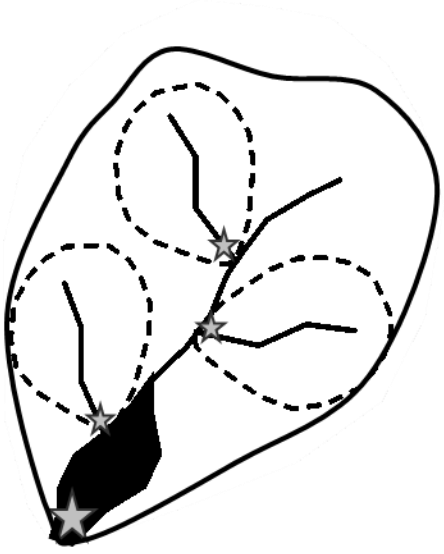


Figure 11. Illustration of assessment points.

SUSTAIN provides the user with a menu of evaluation factors for selection when defining the optimization problem. For example, the optimization objective can be to minimize peak flow, maximize volume controlled, maximize load reduction, or minimize an exceedance frequency of a pollutant concentration. Using these options, a flow target or total maximum daily load (TMDL) water quality target can be formulated as an optimization objective. Table 8 lists the evaluation factor options in *SUSTAIN*.

Table 8. Example control targets for a typical evaluation factor assessment in *SUSTAIN*

Control target	**Target value**		**Note**
Flow			
Peak Discharge	Cubic feet per second	--	Parameters related to increased runoff from urbanization
	Percent reduction of the existing condition	0–100	
	Fraction between existing and pre-developed conditions	0–1	
Annual Average Volume	Cubic feet per year	--	
	Percent reduction of the existing condition	0–100	
	Fraction between existing and pre-developed conditions	0–1	
Exceeding Frequency	Times per year of a given threshold flow rate (cfs)	--	
	Percent reduction of the existing condition	0–100	
	Fraction between existing and pre-developed conditions	0–1	
Sediment			
Annual Average Load	Pounds per year value	––	Parameters to meet the water quality standards or biologically derived parameters to meet designated uses in the waterbody of concern
	Percent reduction of the existing condition	0–100	
	Fraction between existing and pre-developed conditions	0–1	
Annual Average Concentration	Milligram per liter value	--	
	Percent reduction of the existing condition	0–100	
	Fraction between existing and pre-developed conditions	0–1	
Maximum Days Average Concentration	Milligram per liter value of given days	--	
	Percent reduction of the existing condition	0–100	
	Fraction between existing and pre-developed conditions	0–1	
Pollutants (TN, TP, or User Defined)			
Annual Average Load	Pounds per year value	--	Parameters to meet the pollutant criteria (numeric concentration or frequency of exceedance) or TMDL (load allocation) or other locally defined water-protection goals
	Percent reduction of the existing condition	0–100	
	Fraction between existing and pre-developed conditions	0–1	
Annual Average Concentration	Milligram per liter value	--	
	Percent reduction of the existing condition	0–100	
	Fraction between existing and pre-developed conditions	0–1	
Maximum Days Average Concentration	Milligram per liter value of given days	--	
	Percent reduction of the existing condition	0–100	
	Fraction between existing and pre-developed conditions	0–1	

Management Targets

Management targets can be related to either water quality or quantity. The user specifies the water quality or water quantity target value or range for each assessment point.

Optimization Problem Formulations

SUSTAIN provides two optimization options: (1) cost minimization, and (2) cost-effectiveness curve. Option (1) uses the Scatter Search method introduced by Glover (1977), which is a meta-heuristic search technique that has been explored and used in optimizing complex systems (Glover et al. 2000; Laguna and Marti 2002; Zhen et al. 2004). Option (2) uses NSGA-II, which is an advanced genetic algorithm based on Pareto dominance, and uses non-domination and distribution instead of fitness value to score individuals (Deb et al. 2002). In the Scatter Search option, the optimization search process identifies the near-optimal solutions that meet the user-specified management targets. Multiple objectives can be defined during a Scatter Search. With the NSGA-II method, the optimization process reveals all the cost-effective solutions that provide the highest benefit at each cost interval. Only one objective can be defined when generating a cost-effectiveness curve.

Both optimization formulations are defined with the objective of minimizing cost subject to desired water quality or water quantity (or both) objectives at a specified location (assessment point). The optimization problem formulation can be mathematically expressed as below. In the formulation, a group of BMP_i (i = 1,…, n) forms the decision matrix, which defines the optimization engine's search domain. For each potential location, the user defines the feasible range of BMP type and configuration parameters.

The objective is to

$$\text{Minimize } \sum_{i=1}^{n} Cost(BMP_i)$$

subject to

$Q_j \le Qmax_j$ and

$L_k \le Lmax_k$

where

BMP_i = a set of BMP configuration decision variables associated with location i,
Q_j = the computed amount of water quantity factor at the assessment point j,
$Qmax_j$ =the maximum value of the water quantity factor targeted at the assessment point j,
L_k = the computed amount of water quality loading factor at the assessment point k, and
$Lmax_k$ = the maximum value of the water quality loading targeted at the assessment point k.

Optimization Technique – Scatter Search

For the *minimize cost* option, Scatter Search is more efficient (finding the best solutions with fewer runs of the simulation module) than NSGA-II because the search is more focused around the target. The major operation steps of Scatter Search are described below.

Generating a starting set of diverse points (function: InitProblem)

Generating a starting set of diverse solutions which are array of decision variables is accomplished by dividing the range of each variable into four sub-ranges of equal size. Next, a solution is constructed in two steps: a sub-range is first randomly selected and then a value is randomly chosen from the selected sub-range. The starting set of solution points also includes all variables at their lower bound, all variables at their upper bound, all variables at their midpoints, and other solution points suggested by the user.

Choosing a subset of diverse points as the reference set (function: InitRefSet)

The reference set (RefSet), is a collection of both high-quality solutions and diverse solutions that are used to generate new solutions. Specifically, the *RefSet* consists of the union of two subsets, *RefSet1*

and *RefSet2*, of size b_1 and b_2, respectively. That is, |*RefSet*| = $b = b_1 + b_2$. The construction of the initial reference set starts with the selection of the best b_1 solutions from the starting set of diverse points (*P*). The notion of *best* in this step is a measure given by evaluating the objective function. These solutions are added to RefSet and deleted from *P*. For each improved solution in *P* – *RefSet*, the minimum of the Euclidean distances to the solutions in RefSet is computed. Euclidean distance is the straight line distance between two points. For example, in a two-dimensional plane, the Euclidean distance is the straight line between point 1 at (x_1, y_1) and point 2 at (x_2, y_2) and is equal to $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Then, the solution with the maximum of these minimum distances is selected. This solution is added to RefSet and deleted from *P* and the minimum distances are updated. This process is repeated b_2 times. The resulting reference set has b_1 high-quality solutions and b_2 diverse solutions.

Starting the search for the optimal solution by using a linear combination method to construct new solution points from the reference solution points, and Updating the Refset (function: PerformSearch)
The linear combination is based on the three types of formulation, in which x' and x'' are reference solution points, and x_{1-3} is the newly generated solution points:

$x_1 = x' - d$
$x_2 = x'' - d$
$x_3 = x' + d$

where $d = r\frac{x'' - x'}{2}$ and *r* is a random number in the range of (0, 1).

In the course of searching for a global optimum, the *RefSet* is continuously updated. The solutions having better quality, or ones that can improve the diversity of the reference set, replace the old points in the set.

This is a generic description of the scatter search algorithm to illustrate how the new solutions are constructed. In *SUSTAIN*, to improve the search efficiency, the user defines the search increment of each decision variables. Scatter search works with continuous numbers, but if the decision variable has only two states/values (minimum = 0, maximum = 1, and search increment = 1), the new solution is forced to sample only 0 and 1.

Stop the search if the stopping criteria are met
The *stopping* criteria can be defined, at the user's option, either as the maximum number of iteration runs, or the minimum improvement between updates of the reference set, or both, in which case, the search process will be stopped when either of the criteria is met.

Optimization Technique – NSGA-II

Under the NSGA-II cost-effectiveness curve option, the search aims at identifying the cost-effective solutions within the specified management target range. The multi-objective problem can be expressed as follows:

Minimize $\sum_{i=1}^{n} Cost(BMP_i)$ and

Minimize (*EF – Target)*

where

BMP_i = a set of BMP configuration decision variables associated with location *i* and

EF = the management evaluation factor (*EF*) at one given assessment point, and the *EF* can be any of the options listed in Table 8.
Target = the target value of the EF.

For the *cost-effectiveness curve* option, NSGA-II (genetic algorithm) is more efficient than Scatter Search because it applies the non-dominated sorting technique and the search proceeds in a manner of fronts. The major operation steps of NSGA-II are described below.

Creation of first generation
When applying the NSGA-II, a random parent population (P_0) consisting of N solutions is first created. The population is then sorted by the non-dominant level. A solution $x^{(1)}$ is non-dominant to another solution when $x^{(1)}$ performs no worse than the other solution in all objectives, and $x^{(1)}$ performs better than the other solution in at least one objective. At the end of the sorting, each solution is assigned a fitness (or rank) equal to its non-dominant level, with a smaller value indicating that the solution is dominated by fewer other solutions.
The processes of tournament selection, crossover, and mutation are used to create a child population (Q_0), which has a same size of P_0 with N solutions.

Main loop
In the first step of the main loop, the parent population and the child population are combined (R_0). The population of R_0 will have $2N$ solutions. The $2N$ solutions in R_0 are then sorted according to non-domination. Elitism is ensured in this step because both the parent and the child population are used in the sorting. The sorted $2N$ solutions will form various best non-dominated subsets (in which all the solutions are non-dominant to each other, but overall they dominate other subsets). The first N solutions from the ranked best non-dominant subsets, F_1,…, F_i, are then selected to form a new parent population (P_1). The new parent population is used to create a new child population (O_1), and the process continues until the stopping criteria are met.

The NSGA-II uses the crowding distance (the size of the largest cuboid enclosing solution $x^{(1)}$ without including any other solution in the population) concept to maintain solution diversity. That is, in cases where two solutions have the same non-domination rank, the solution with larger crowding distance is always preferred.

Stopping criteria
The user can make the NSGA-II stop when the new parent population does not change for two consecutive loops. The stopping criterion can also be that the fitness function does not improve after a certain number of iterations.

Tiered Optimization

SUSTAIN is able to evaluate management practices at multiple scales, ranging from local to watershed applications. Management plans often need to evaluate the cumulative benefit of management practices at multiple- scale watersheds on downstream water quality in rivers, lakes, or estuaries. The site or local-scale evaluation involves simulation and analyses of individual BMPs and various combinations of practices and treatment trains to derive local runoff quantity and quality. For a larger-scale watershed, there could be hundreds or thousands of individual management practices that are implemented to achieve a desired cumulative benefit. *SUSTAIN* incorporates an innovative, tiered approach that allows for cost-effectiveness evaluation of both individual and multiple nested watersheds to address the needs of both regional and local-scale applications (Figure 12).

A relatively large watershed can usually be subdivided into several smaller subwatersheds as shown in Figure 12. Users need to select, with, say, the use of the siting tool in *SUSTAIN*, an appropriate suite of

feasible BMP options (types, configurations, and costs) at strategic locations for each subwatershed. *SUSTAIN* then generates the time series rainfall-runoff data from BMP drainage areas and routes them through BMPs, in parallel or in series, to produce the quantity and quality data at downstream assessment points. *SUSTAIN* uses the cost and effectiveness data to derive the cost-effectiveness curve that relates flow or pollutant-load reductions with costs. Each point on the cost-effectiveness curve represents an optimal combination of BMPs that will collectively remove the targeted amount of pollutant load at the least cost.

The tiered optimization procedures implemented in *SUSTAIN* provides an efficient and manageable means for large-scale applications and allows users to evaluate and optimize on the basis of the hydrologic and water quality characteristics at the specified assessment points. Tier-1 performs the optimization search to develop cost-effectiveness curves for each tier-1 subwatershed. In a tier-2 analysis, the tier-1 solutions are used to construct a new optimization search domain and run the transport module, if needed, with solutions from all the tier-1 subwatersheds to develop the combined cost-effectiveness curve for the entire watershed.

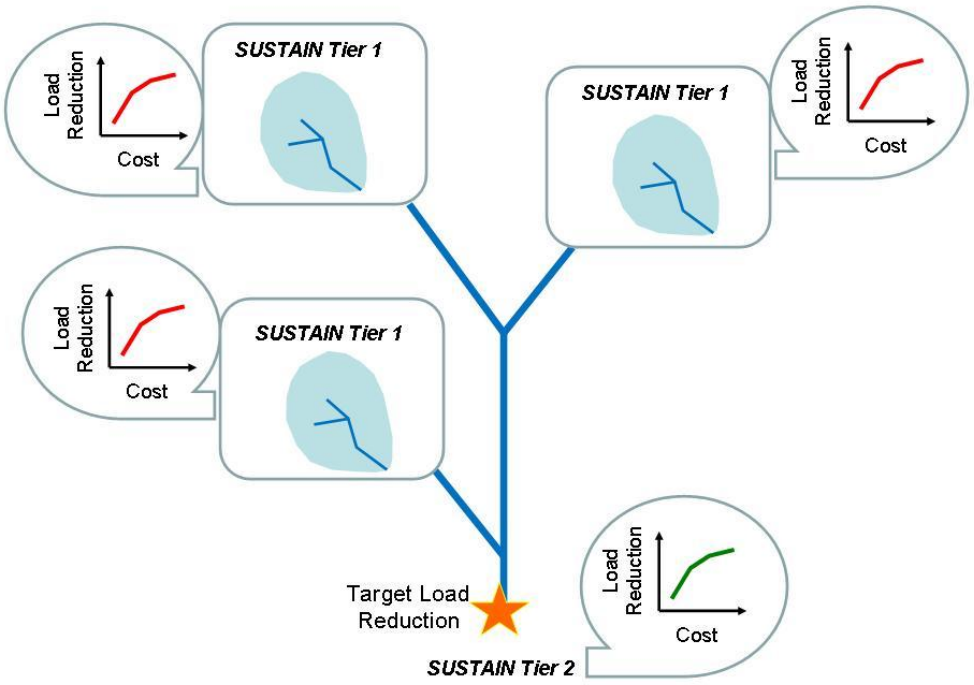


Figure 12. Tiered application of *SUSTAIN* for developing cost-effectiveness curves.

Figure 13 illustrates the tiered application process in more detail. At the first step (tier-1) of the tiered optimization analysis, the cost-effectiveness curve for each subwatershed is generated by performing continuous multiple optimization runs at incremental flow/pollutant reduction targets. In the second step (tier-2), the search domain is constructed using the tier-1 results. As shown, the search domain for tier-2 contains the discrete solutions on the tier-1 cost-effectiveness curves at assessment points i and j. The third step is to perform the tier-2 optimization for the search domain constructed. The optimization engine strategically samples the discrete options in the search domain. The cost-effectiveness of each sample is measured, stored, and analyzed to guide the next search direction.

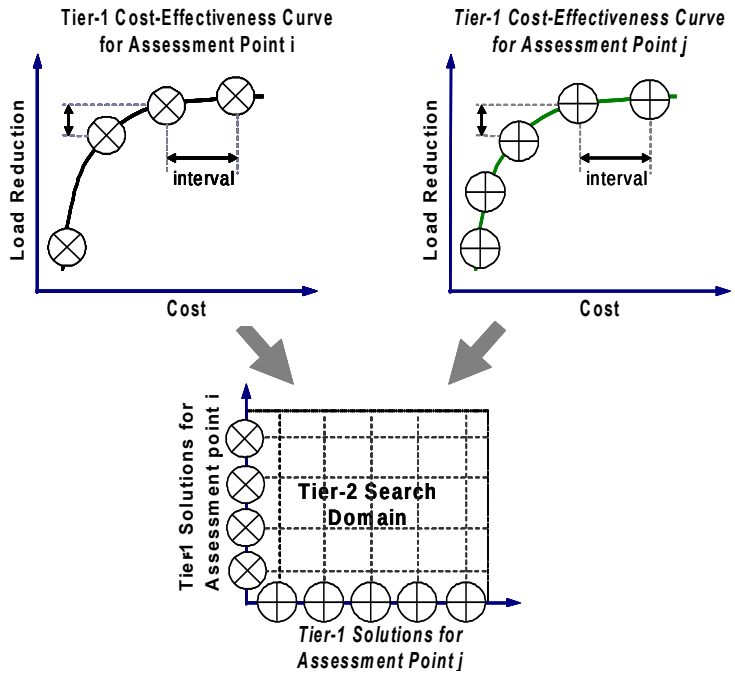


Figure 13. Construction of the tier-2 search domain using tier-1 results.

Figure 14 illustrates the simulation process used to generate the results for measuring the cost-effectiveness of each iteration in the tier analysis. The simulated time series outputs for all discrete points on the tier-1 cost-effectiveness curve are stored and used when a point, hence the BMP options associated with it, is chosen in the tier-2 analysis. Similarly, the time series runoff data of the watershed area that is not part of the tributary areas of the tier-1 assessment points is generated and stored before the tiered analysis. This data, however, might also be generated during the tier-2 search process. The transport module is often required to perform routing of the time series data from the upstream tier-1 subwatershed to merge with that for the downstream tier-1 subwatershed. In such a manner, the tiered approach is applied to a large watershed which contains subwatersheds or to a small watershed that requires the development of a detailed management plan at a parcel- or a street-block-level.

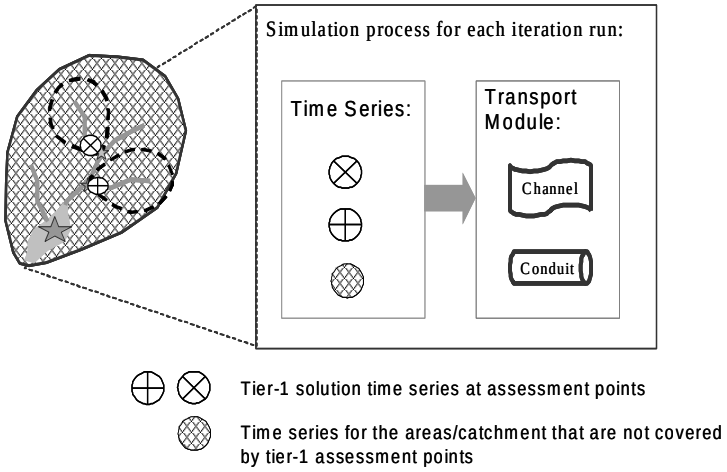


Figure 14. Simulation process for each iteration run.

SUSTAIN Simulation Engine Project

This section provides the software requirements and the properties settings for Visual C++ project of *SUSTAIN* simulation engine. The required software is Miscrosoft Visual C++ program, which is part of the Visual Studio 6.0 package. Microsoft Visual Studio 6.0 is compatible with Windows XP and might not be supported by any latest Windows operating system. The simulation engine uses the Microsoft Foundation Class library to use many of the available functionalities in that library and, therefore, needs to be specified in the project settings as shown inFigure 15. Figure 15 also shows the output file name of the dynamic link library for the simulation engine (i.e., SUSTAINOPT.dll). Figure 16 shows the project options for defining the directory paths of source and library files used in compiling the source code. The project settings and options are required only once before compiling the source code for the first time.

Software Requirements

- Microsoft Visual C++ (Visual Studio 6.0)
- Operating System: Windows XP

Project Properties

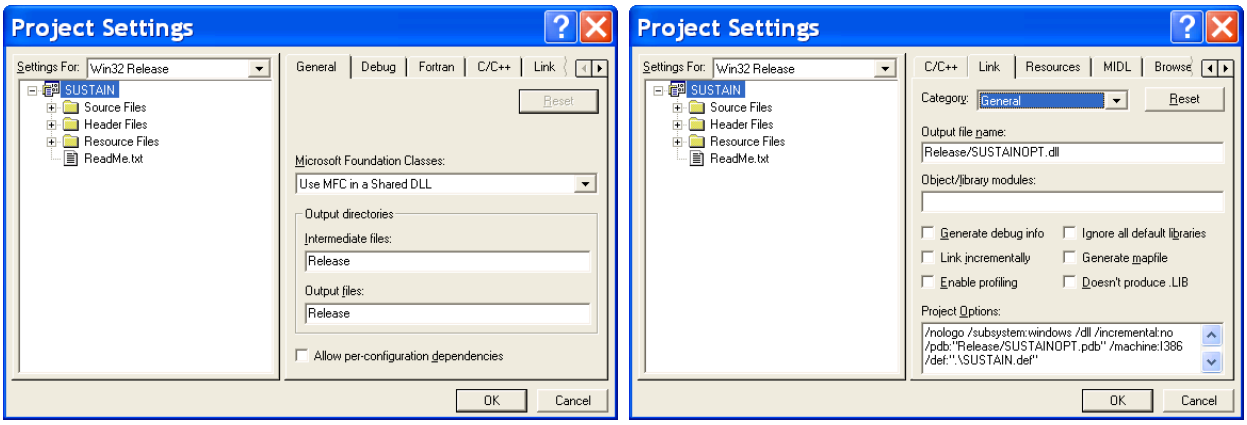


Figure 15. Screenshots of *SUSTAIN* Visual C++ project settings.

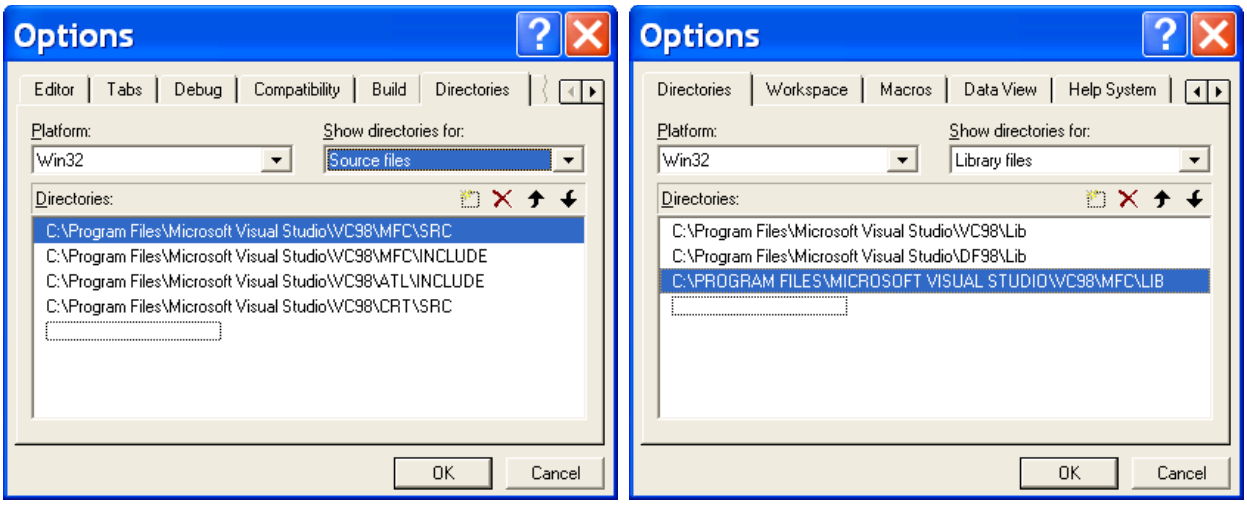


Figure 16. Screenshots of *SUSTAIN* Visual C++ project options.

Data Flow Model

The *SUSTAIN* simulation engine is called from the model's ArcGIS interface. The ArcGIS interface loads the GIS database and BMP cost database to populate the user input screens for defining BMP templates, placing BMPs on the map, and defining the BMP routing network. The data flow model for the simulation engine is shown in Figure 17. The ArcGIS interface launches the post-processor for viewing the model results once the simulation engine finishes the optimization runs.

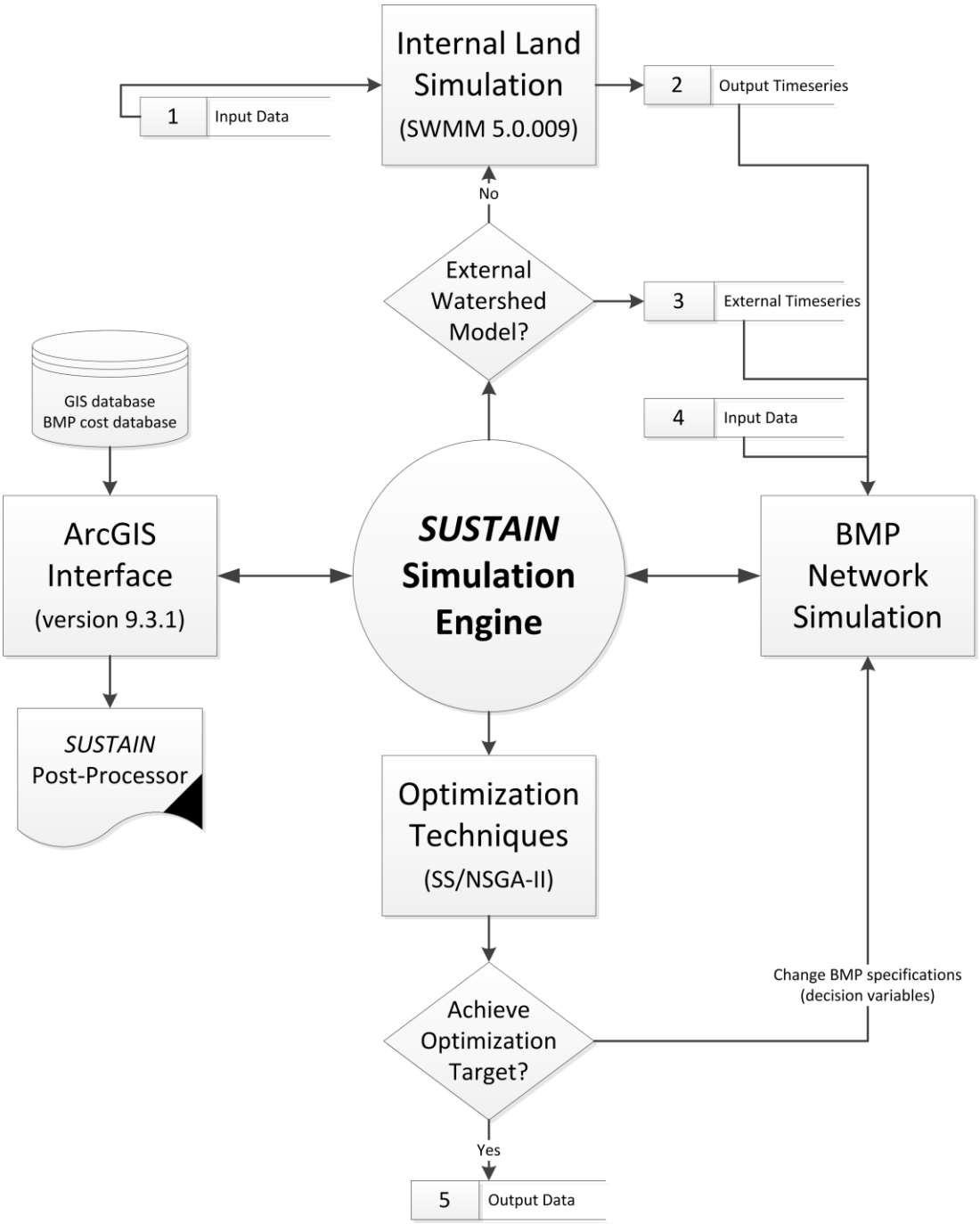


Figure 17. *SUSTAIN* simulation data flow diagram.

The list of data files used in *SUSTAIN* simulation engine is shown below.

1. Input data for internal land simulation (1: Input Data in Figure 17): This dataset is optional if the user selects to use the external land use time series option in *SUSTAIN*. The dataset for internal land simulation option includes the following:
   - Climate time series file: The land simulation module of *SUSTAIN* uses the daily air temperature, evaporation, and wind speed data from the user-specified climate file. The format for climate file is consistent with that used in the SWMM, where each line in the file contains a recording station name, year, month, day, maximum temperature, minimum temperature, and, optionally, the evaporation rate and wind speed. The data must be in U.S. units: temperature in degrees F, evaporation in in./day, and wind speed in mi/hr, all separated by one or more spaces.
   - Precipitation time series file: The precipitation data is input in a separate file where each line of the file contains the station ID, year, month, day, hour, minute, and precipitation, all separated by one or more spaces. The precipitation data type in any one of these three formats: (1) intensity, where the value is an average rate (in./hr) over the recording interval; (2) interval volume, where the value is the volume of rain that fell in the recording interval (in.); or (3) cumulative volume, where the value represents the cumulative rainfall that has occurred since the start of the last series of non-zero values (in.).
   - The input text files: The ArcGIS interface of *SUSTAIN* creates two input text files for the EPA SWMM to simulate the land compartment. One file is for the current land use scenario, and the other one is for the user-specified, predeveloped single land use scenario.
2. Output time series from internal land simulation (2: Output Timeseries in Figure 17): The land simulation module in *SUSTAIN* outputs the aggregated land response (i.e., hydrograph and pollutograph) at each node of the BMP network (BMP or Junction) defined in the *SUSTAIN* project. The output data include the following:
   - The output time series files (ASCII format): The internal land simulation module creates two time series files. One file is for the current land use scenario, and the other one is for user-specified, predeveloped land use scenario. The format for these files is consistent with that used in the SWMM, where each line in the file contains a node ID, year, month, day, hour, minute, second, the outflow rate (cfs), and the concentration for each pollutant (mg/L).
   - The output time series files (binary format): These files (the current land use scenario and the predeveloped land use scenario) are similar to the above mentioned time series files but are in binary format and are not used in *SUSTAIN*. Those files can be used with EPA SWMM (external to *SUSTAIN* model) to view the land output results. *SUSTAIN* does not provide any utility tool to view or process the land output time series.
   - The output summary text files: The summary text files (the current land use scenario and the predeveloped land use scenario) summarize the SWMM results and report error messages if errors occur in the model run.
3. External time series for land simulation (3: External Timeseries in Figure 17): This dataset is optional if the user selects to use the internal land use simulation option in *SUSTAIN*. The externally generated time series represent hydrology and water quality at the landscape level. The external option in *SUSTAIN* allows importation of the hydrograph and pollutograph for each land use category (or combination of land use, slope, and hydrologic soil group) from an external watershed model such as HSPF or LSPC model. It uses the sub-hourly (1–60 min) flow and pollutant loading data from the user-specified time series file. The format for the time series file is consistent with that output from the HSPF or LSPC model, where each line in the file contains a watershed ID (dummy value not used in *SUSTAIN*), year, month, day, hour, minute, flow volume (in.-acre), groundwater recharge volume (in.-acre), and the loads for each pollutant per unit area (acre). The data must be in U.S. units, separated by space or tab delimiters.

4. Input data for BMP network simulation (4: Input Data in Figure 17): This dataset is in addition to the external/internal land use time series data mentioned in the previous steps 2 and 3. These additional data include the following:
   - Climate time series file (optional): The BMP simulation module of *SUSTAIN* uses the daily air temperature and evaporation data from the user-specified climate file if specified in the input text file. The format for climate file is consistent with that used in the SWMM, where each line in the file contains a recording station name, year, month, day, maximum temperature, minimum temperature, and the evaporation rate. The data must be in U.S. units: temperature in degrees F, and evaporation in in./day, all separated by one or more spaces.
   - Pont source time series file (optional): The point source data can be input at any node of BMP network (BMP or Junction). It uses the sub-hourly (1–60 min) flow and pollutant loading data from the user-specified time series file. The format for the time series file is consistent with the external watershed model time series, where each line in the file contains a point source ID (dummy value not used in *SUSTAIN*), year, month, day, hour, minute, flow volume (in.-acre), and the loads for each pollutant. The data must be in U.S. units, separated by space or tab delimiters. Note that the point source time series file does not have a groundwater recharge column as compared to the external time series file.
   - The input text file: The ArcGIS interface of *SUSTAIN* creates an input text file for BMP network simulation. The file contains the file path for the land output (internal/external) time series, BMP specifications, BMP network routing information, and optimization parameters.
5. Output data at the assessment point (5: Output Data in Figure 17): The *SUSTAIN* simulation engine outputs the model results at the user-specified assessment point (BMP or Junction). The output data include the following:
   - Hydrology and water quality time series files: The *SUSTAIN* model generates output time series files for Initial Condition (no optimization), Pre-Development, Post-Development, and Best-Solution (optimized) scenarios at each assessment point. The data in the time series file are in tab-delimited format, where each line in the file contains assessment point ID (BMP or Junction), year, month, day, hour, minute, BMP volume (ft^3), water depth (ft), total inflow (cfs), weir outflow (cfs), orifice or channel outflow (cfs), underdrain outflow (cfs), untreated (bypass) outflow (cfs), total outflow (cfs), surface infiltration (cfs), percolation from soil media to underdrain storage (cfs), total evapotranspiration (cfs), seepage to groundwater (cfs), and the total mass entering the BMP (lbs), mass leaving (weir outflow) the BMP (lbs), mass leaving (orifice outflow) the BMP (lbs), mass leaving (underdrain outflow) the BMP (lbs), mass bypassing (untreated) the BMP (lbs), total mass leaving the BMP (lbs), total outflow concentration (mg/L) for each pollutant. The time series file also contains a header section describing the output parameters in the order they are reported in the file. If the user runs a best solution scenario on the cost-effectiveness curve from the spreadsheet post-processor, the model generates a similar time series file labeled as BestPop with the solution ID shown on the curve.
   - Evaluation summary files: The model outputs an evaluation summary file for Initial Condition (no optimization), Pre-Development, Post-Development, and Best-Solution (optimized) scenarios at each assessment point. The data in the summary file are in tab-delimited format, where each line in the file contains assessment point ID (BMP or Junction), evaluation factor type (such as average annual flow volume), evaluation factor value, and the total project cost.
   - Optimization summary files: The model outputs two summary files for the optimization runs, AllSolutions and BestSolutions. The data in each summary file are in tab-delimited format, where each line in the file contains the solution ID, total project cost, total surface area of the selected BMPs, total excavation volume of the selected BMPs, total surface

storage volume of the selected BMPs, total soil storage volume of the selected BMPs, total underdrain storage volume of the selected BMPs, value for each evaluation factor, total cost for each selected BMP type, and the selected value for the decision variables (such as BMP sizes) in this solution. If the user runs a best solution scenario on the cost-effectiveness curve from the spreadsheet post-processor, the model generates a summary file labeled as CECurve_Solutions that contains the sequence number for the selected best population and solution, total project cost, value for each evaluation factor, and the selected value for the decision variables (such as BMP sizes) in that solution.

Class Documentation

The section describes the functions and variables used in different classes, and it identifies all relevant links to the classes, member functions, and member variables. A standard program, Doxygen (http://www.stack.nl/~dimitri/doxygen/index.html), for generating technical documentation from source code files was used to create the essential elements of this document. The program uses the comments blocks, which are written in a required format, to document the functions and variables used in *SUSTAIN*.

Microsoft Foundation Class References

The following class objects are used from the Microsoft Foundation Class (MFC) Library in this program.

- CString: A CString object represents a sequence of a variable number of characters and can be thought of as arrays of characters. This class provides support for manipulating strings and extends the functionality normally provided by the C runtime library string package. The CString class provides member functions and operators for simplified string handling. This also provides constructors and operators for constructing, assigning, and comparing CString data types.
- COleDateTime: A COleDateTime value represents an absolute date and time value that encapsulates the DATE data type. The DATE type is implemented as a floating-point value, measuring days from midnight. This class handles dates from January 1, 100 – December 31, 9999 and does not support Julian dates.
- CObList: A CObList class supports list of CObject pointers accessible sequentially or by pointer value. A variable of type POSITION is used as an iterator to navigate a list sequentially. Adding an element is very fast at the list head, at the tail, and at a known POSITION. A sequential search is necessary to look up an element by value or index, and it can be slow if the list is long. When a CObList object is deleted, or when its elements are removed, only the CObject pointers are removed, not the objects they reference.
- CPtrList: A CPtrList class supports lists of void pointers. The member functions of CPtrList are similar to the member functions of class CObList. When a CPtrList object is deleted, or when its elements are removed, only the pointers are removed, not the entities they reference.

ADJUSTABLE_PARAM Struct Reference

#include "BMPSite.h"

Public Attributes

- CString m_strVariable
- double m_lfFrom
- double m_lfTo
- double m_lfStep

Detailed Description

This is the data structure class for BMP decision variables. The decision variables are the BMP properties (length, soil depth, etc.) that are allowed to vary during the course of an optimization run with in the user-specified range and increment.

Member Data Documentation

double ADJUSTABLE_PARAM::m_lfFrom

This is the minimum value of the decision variable.

double ***ADJUSTABLE_PARAM::m_lfStep***

This is the increment value of the decision variable.

double ***ADJUSTABLE_PARAM::m_lfTo***

This is the maximum value of the decision variable.

CString ***ADJUSTABLE_PARAM::m_strVariable***

This is the name of the BMP decision variable.

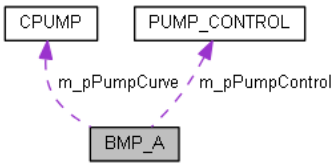
The documentation for this struct was generated from the following file:

- BMPSite.h

BMP_A Struct Reference

#include "BMPSite.h"

Collaboration diagram for BMP_A:



Public Attributes

- int m_nExitType
- int m_nWeirType
- int m_nORelease
- int m_nPeople
- int m_nDays
- double m_lfBasinWidth
- double m_lfBasinLength
- double m_lfOrificeHeight
- double m_lfOrificeDiameter
- double m_lfOrificeCoeff
- double m_lfWeirHeight
- double m_lfWeirWidth
- double m_lfWeirAngle
- double m_lfPETmultiplier
- double m_lfWatDep_i
- double m_lfTheta_i
- double m_lfRelease [24]
- TGrnAmpt m_pGAInfil
- THorton m_pHortonInfil
- PUMP_CONTROL m_pPumpControl
- CPUMP * m_pPumpCurve

Detailed Description

This is the data structure class for BMP Type A. The supported BMPs in *SUSTAIN* under this type are Bioretention, Cistern, Constructed Wetland, Green Roof, Infiltration Basin, Infiltration Trench, Porous Pavement, Rain Barrel, Sand Filter (non-surface), Sand Filter (surface), Wet Pond, Dry Pond, and Regulator.

Member Data Documentation

*double **BMP_A::m_lfBasinLength***

This is the BMP length (ft).

*double **BMP_A::m_lfBasinWidth***

This is the BMP width (ft).

*double **BMP_A::m_lfOrificeCoeff***

This is the orifice discharge coefficient for the orifice equation.

*double **BMP_A::m_lfOrificeDiameter***

This is the orifice diameter (ft).

***double* BMP_A::m_lfOrificeHeight**

This is the orifice height (ft) from the bed.

***double* BMP_A::m_lfPETmultiplier**

This is the PET multiplier.

***double* BMP_A::m_lfRelease[24]**

This is the hourly water release per capita from the cistern control (ft^3/hr/capita).

***double* BMP_A::m_lfTheta_i**

This is the initial soil moisture that identifies antecedent soil moisture conditions (ft/ft).

***double* BMP_A::m_lfWatDep_i**

This is the initial water depth that identifies standing water on surface from antecedent conditions (ft).

***double* BMP_A::m_lfWeirAngle**

This is the weir angle in degrees (for Triangular type).

***double* BMP_A::m_lfWeirHeight**

This is the weir height (ft).

***double* BMP_A::m_lfWeirWidth**

This is the weir width (ft).

***int* BMP_A::m_nDays**

This is the number of dry days after which the stored water is released (for Rain Barrel only).

***int* BMP_A::m_nExitType**

This is the orifice shape type.

***int* BMP_A::m_nORelease**

This is the release type (1-Cistern, 2-Rain Barrel, 3-Orifice).

***int* BMP_A::m_nPeople**

This is the number of people that use the stored water (for Cistern only).

***int* BMP_A::m_nWeirType**

This is the weir type (1-Rectangular and 2-Triangular).

***TGrnAmpt* BMP_A::m_pGAInfil**

This is the data structure for Green-Ampt infiltration parameters.

***THorton* BMP_A::m_pHortonInfil**

This is the data structure for Horton infiltration parameters.

***PUMP_CONTROL* BMP_A::m_pPumpControl**

This is the data structure type of PUMP_CONTROL class.

***CPUMP ** BMP_A::m_pPumpCurve**

This is the pointer to the CPUMP class.

The documentation for this struct was generated from the following file:

- BMPSite.h

BMP_B Struct Reference

#include "BMPSite.h"

Public Attributes

- double m_lfBasinWidth
- double m_lfBasinLength
- double m_lfMaximumDepth
- double m_lfSideSlope1
- double m_lfSideSlope2
- double m_lfSideSlope3
- double m_lfManning
- double m_lfPETmultiplier
- double m_lfWatDep_i
- double m_lfTheta_i
- TGrnAmpt m_pGAInfil
- THorton m_pHortonInfil

Detailed Description

This is the data structure class for BMP Type B. The supported BMP in *SUSTAIN* under this type is Grassed Swale.

Member Data Documentation

double ***BMP_B::m_lfBasinLength***

This is the BMP length (ft).

double ***BMP_B::m_lfBasinWidth***

This is the BMP width (ft).

double ***BMP_B::m_lfManning***

This is the Manning's roughness coefficient used in Manning's equation.

double ***BMP_B::m_lfMaximumDepth***

This is the BMP maximum depth (ft).

double ***BMP_B::m_lfPETmultiplier***

This is the PET multiplier.

double ***BMP_B::m_lfSideSlope1***

This is the BMP side slope for trapezoidal cross-section (ft/ft).

double ***BMP_B::m_lfSideSlope2***

This is the BMP side slope for trapezoidal cross-section (ft/ft).

double ***BMP_B::m_lfSideSlope3***

This is the BMP longitudinal slope (ft/ft).

double ***BMP_B::m_lfTheta_i***

This is the initial soil moisture that identifies antecedent soil moisture conditions (ft/ft).

double ***BMP_B::m_lfWatDep_i***

This is the initial water depth that identifies standing water on surface from antecedent conditions (ft).

TGrnAmpt ***BMP_B::m_pGAInfil***

This is the data structure for Green-Ampt infiltration parameters.

THorton ***BMP_B::m_pHortonInfil***

This is the data structure for Horton infiltration parameters.

The documentation for this struct was generated from the following file:

- BMPSite.h

BMP_C Struct Reference

#include "BMPSite.h"

Public Attributes

- int m_nIndex
- CString m_strID
- CString m_strCondType
- CString m_strCondName
- TLink m_pTLink
- TConduit m_pTConduit
- TTransect m_pTTransect

Detailed Description

This is the data structure class for BMP Type C. A Conduit (open channel and pipes) in *SUSTAIN* is an example of this type.

Member Data Documentation

int ***BMP_C::m_nIndex***

This is the BMP index in the conduit object list.

TConduit ***BMP_C::m_pTConduit***

This is the data structure for conduit parameters.

TLink ***BMP_C::m_pTLink***

This is the data structure for link parameters.

TTransect ***BMP_C::m_pTTransect***

This is the data structure for transect parameters.

CString ***BMP_C::m_strCondName***

This is the name of conduit for an irregular cross-section.

CString ***BMP_C::m_strCondType***

This is the shape type of the conduit cross-section.

CString ***BMP_C::m_strID***

This is the unique identifier of the conduit.

The documentation for this struct was generated from the following file:

- BMPSite.h

BMP_D Struct Reference

#include "BMPSite.h"

Public Attributes

- CString m_strID
- int m_nPolRemMethod
- float m_lfN
- float m_lfdStore
- float m_lfalpha
- float m_lfinflow
- float m_lfrunoff
- float m_lfdepth
- float m_lflosses
- double m_lfBufferWidth
- double m_lfFlowLength
- double m_lfPETmultiplier
- double m_lfWatDep_i
- double m_lfTheta_i
- double m_lfOverlandSlope
- TGrnAmpt m_pGAInfil
- THorton m_pHortonInfil

Detailed Description

This is the data structure class for BMP Type D. A Filterstrip in *SUSTAIN* is an example of this type.

Member Data Documentation

float ***BMP_D::m_lfalpha***

This is the overland flow factor used in the nonlinear reservoir routing equation.

double ***BMP_D::m_lfBufferWidth***

This is the buffer width perpendicular to the surface runoff (ft).

float ***BMP_D::m_lfdepth***

This is the depth of surface runoff (ft).

float ***BMP_D::m_lfdStore***

This is the depression storage on the surface (ft).

double ***BMP_D::m_lfFlowLength***

This is the buffer length along the surface runoff (ft).

float ***BMP_D::m_lfinflow***

This is the inflow rate (ft/sec).

float ***BMP_D::m_lflosses***

This is the evaporation and infiltration loss rate (ft/sec).

float ***BMP_D::m_lfN***

This is the Manning's roughness coefficient.

double ***BMP_D::m_lfOverlandSlope***

This is the overland slope (ft/ft).

double ***BMP_D::m_lfPETmultiplier***

This is the PET multiplier to reflect the vegetation cover.

float ***BMP_D::m_lfrunoff***

This is the runoff rate (ft/sec).

double ***BMP_D::m_lfTheta_i***

This is the initial soil moisture that identifies antecedent soil moisture conditions (ft/ft).

double ***BMP_D::m_lfWatDep_i***

This is the initial water depth that identifies standing water on the surface from antecedent conditions (ft).

int ***BMP_D::m_nPolRemMethod***

This is the pollutant removal method (0-1st order decay, 1-Kadlec and Knight method).

TGrnAmpt ***BMP_D::m_pGAInfil***

This is the data structure for Green-Ampt infiltration parameters.

THorton ***BMP_D::m_pHortonInfil***

This is the data structure for Horton infiltration parameters.

CString ***BMP_D::m_strID***

This is the unique identifier of the BMP.

The documentation for this struct was generated from the following file:

- BMPSite.h

BMP_E Struct Reference

#include "BMPSite.h"

Public Attributes

- CString m_strID
- int m_nPolRemMethod
- float m_lfN
- float m_lfdStore
- float m_lfalpha
- float m_lfinflow
- float m_lfrunoff
- float m_lfdepth
- float m_lflosses
- float m_lfsArea
- float m_lfSatInfil
- double m_lfDCIA
- double m_lfFlowLength
- double m_lfOverlandSlope
- double m_lfTotImpArea

Detailed Description

This is the data structure class for BMP Type E. An Impervious Area Disconnection (Area BMP) in *SUSTAIN* is an example of this type.

Member Data Documentation

float ***BMP_E::m_lfalpha***

This is the overland flow factor.

double ***BMP_E::m_lfDCIA***

This is the percent of a directly connected impervious area (0-100).

float ***BMP_E::m_lfdepth***

This is the depth of surface runoff (ft).

float ***BMP_E::m_lfdStore***

This is the surface depression storage (ft).

double ***BMP_E::m_lfFlowLength***

This is the buffer length along the surface runoff (ft).

float ***BMP_E::m_lfinflow***

This is the inflow rate (ft/sec).

float ***BMP_E::m_lflosses***

This is the evaporation and infiltration loss rate (ft/sec).

float ***BMP_E::m_lfN***

This is the Manning's roughness coefficient.

double ***BMP_E::m_lfOverlandSlope***

This is the overland slope (ft/ft).

float ***BMP_E::m_lfrunoff***

This is the runoff rate (ft/sec).

float ***BMP_E::m_lfsArea***

This is the BMP surface area (ft^2).

float ***BMP_E::m_lfSatInfil***

This is the saturated infiltration rate (ft/sec).

double ***BMP_E::m_lfTotImpArea***

This is the total impervious area (acre).

int ***BMP_E::m_nPolRemMethod***

This is the pollutant removal method (0-1st order decay, 1-Kadlec and Knight method).

CString ***BMP_E::m_strID***

This is the unique identifier of the BMP.

The documentation for this struct was generated from the following file:

- BMPSite.h

BMPCOST Struct Reference

#include "BMPData.h"

Public Attributes

- int m_nBMPClass
- CString m_strBMPType
- double m_lfCost

Detailed Description

This is the BMP cost data structure class.

Member Data Documentation

double ***BMPCOST::m_lfCost***

This is the total cost for the same BMP types.

int ***BMPCOST::m_nBMPClass***

This is the unique identifier for the BMP type (1-class A, 2-class B, 3-class C, 4-class D, 5-class E).

CString ***BMPCOST::m_strBMPType***

This is the descriptive name for the BMP type.

The documentation for this struct was generated from the following file:

- BMPData.h

CAquifer Class Reference

#include "Aquifer.h"

Public Member Functions

- CAquifer ()
- CAquifer (int nAquiferID, CString strAquiferName, double lfGWrc, double lfGWsc, double lfGWStorage)
- virtual ~CAquifer ()

Public Attributes

- int m_nAquiferID
- double m_lfGWrc
- double m_lfGWsc
- double m_lfGWStorage
- double * m_pConc
- CString m_sAquiferName

Detailed Description

This is the data structure for the CAquifer class. The aquifer system in *SUSTAIN* is supported in the External Land Simulation option only. An aquifer receives water from the groundwater recharge provided in the external land time series and the background infiltration in BMPs. An aquifer can receive water from more than one land use or BMP, but it can release the water to only one conduit.

Constructor & Destructor Documentation

CAquifer::CAquifer ()

This is the CAquifer class constructor (default).

CAquifer::CAquifer (int nAquiferID*, CString* strAquiferName*, double* lfGWrc*, double* lfGWsc*, double* lfGWStorage*)*

This is the CAquifer class constructor that initializes the following class parameters.

Parameters:

nAquiferID	Aquifer unique ID (non zero).
strAquiferName	Aquifer descriptive name.
lfGWrc	Groundwater recession coefficient (per hour).
lfGWsc	Groundwater seepage coefficient (per hour).
lfGWStorage	Groundwater stored volume (ac-ft).

CAquifer::~CAquifer () [virtual]

This is the CAquifer class destructor.

Member Data Documentation

double CAquifer::m_lfGWrc

This is the groundwater recession coefficient used to release water to a conduit (per hour).

double CAquifer::m_lfGWsc

This is the groundwater seepage coefficient used to calculate the deep percolation (loss) from the system (per hour).

double CAquifer::m_lfGWStorage

This is the available stored volume in the aquifer system (ac-ft).

int ***CAquifer::m_nAquiferID***

This is the aquifer's unique ID (non zero).

*double ** ***CAquifer::m_pConc***

It is an array of pollutants concentration in the aquifer (mg/L).

CString ***CAquifer::m_sAquiferName***

This is the aquifer's descriptive name.

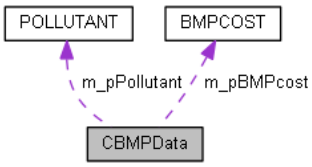
The documentation for this class was generated from the following files:

- Aquifer.h
- Aquifer.cpp

CBMPData Class Reference

#include "BMPData.h"

Collaboration diagram for CBMPData:



Public Member Functions

- CBMPData ()
- virtual ~CBMPData ()
- CBMPSite * FindBMPSite (const CString &strID)
- CLandUse * FindLandUse (int nLuID)
- CAquifer * FindAquifer (int nAquiferID)
- CPUMP * FindPumpCurve (CString strPCurveID)
- void SkipCommentLine (FILE *fp)
- void OutputFileHeaderForTradeOffCurve (FILE *fp)
- void WriteFileHeader (FILE *fp, int NWQ)
- void AddRouteNode (CBMPSite *pBMPSite)
- bool ReadInputFile (CString strFileName)
- bool ReadFileSection (FILE *fp, int nSection)
- bool ReadDataLine (FILE *fp, CString &strData)
- bool ReadBestPopFile (int nBestPopId)
- bool RoutingCycleExist (CBMPSite *pBMPSite)
- bool PrepareDataForModel ()
- bool OpenOutputFiles (const CString &runID)
- bool CloseOutputFiles ()
- bool ProcessPollutantData ()
- bool ProcessTransportData ()
- bool LoadClimateTSData (COleDateTime startDate, COleDateTime endDate)

Public Attributes

- CString strInputDir
- CString strOutputDir
- CString strMixLUFileName
- CString strPreLUFileName
- CString strError
- CString strClimateFileName
- COleDateTime startDate
- COleDateTime endDate
- int m_nSedQualFlag
- int nStrategy
- int nLandSimulation
- int nLANDTimeStep
- int nBMPTimeStep
- int nOutputTimeStep
- int nBIORETENTION
- int nWETPOND
- int nCISTERN
- int nDRYPOND
- int nINFILTRATIONTRENCH
- int nGREENROOF

- int nPOROUSPAVEMENT
- int nRAINBARREL
- int nREGULATOR
- int nSWALE
- int nBUFFERSTRIP
- int nAREABMP
- int nBMPtype
- int nBMPA
- int nBMPB
- int nBMPC
- int nBMPD
- int nBMPE
- int nAdjVariable
- int nEvalFactor
- int nRunOption
- int nSolution
- int nPollutant
- int nNWQ
- int m_nNum
- int nETflag
- int * nSedflag
- double lfStopDelta
- double lfMaxRunTime
- double lfLatitude
- double lfmonET [12]
- double * m_pDataClimate
- double * polmultiplier
- CObList luList
- CObList siteluList
- CObList bmpsiteList
- CObList routeList
- CObList sitepsList
- CObList AquiferList
- CObList PumpList
- POLLUTANT * m_pPollutant
- BMPCOST * m_pBMPcost

Detailed Description

This is the Data Management class in *SUSTAIN* that defines the data structure for the model input and output parameters. This class populates the data structures (arrays, variables, objects pointers, etc.) while reading the BMP optimization input text file. The member functions of this class open the model output file, write the model output, and process the input data for the model simulation run.

Constructor & Destructor Documentation

CBMPData::CBMPData ()

This is the CBMPData class constructor (default).

CBMPData::~CBMPData () [virtual]

This is the CBMPData class destructor.

Member Function Documentation

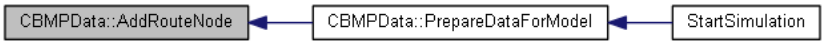
*CBMPData::AddRouteNode (CBMPSite * pBMPSite)*

This function adds the given BMP to the routing list of BMP sites in the correct sequence for the model simulation run.

Parameter:

pBMPSite	The pointer to the given BMP site.

This is the caller graph for this function:



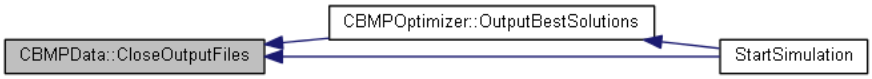
CBMPData::CloseOutputFiles ()

This function closes the model output time series file.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:



CBMPData::FindAquifer (int nAquiferID)

This function searches for the given aquifer in the AquiferList.

Parameter:

nAquiferID	Unique identifier for the given aquifer.

Returns:

Null if it fails, else pointer to the given aquifer of CAquifer class type.

This is the caller graph for this function:

CBMPData::FindBMPSite (const CString & strID)

This function searches for the given BMP site in the bmpsiteList.

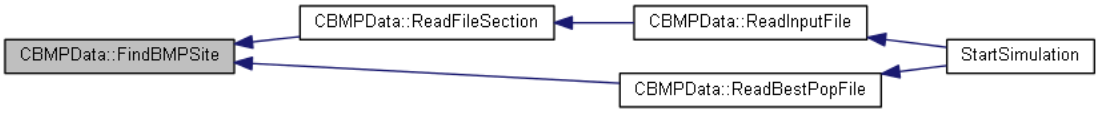
Parameter:

strID	Unique identifier for the given BMP site.

Returns:

Null if it fails, else pointer to the given BMP site of CBMPSite class type.

This is the caller graph for this function:



CBMPData::FindLandUse (int nLuID)

This function searches for the given land use in the luList.

Parameter:

nLuID	Unique identifier for the given land use.

Returns:

Null if it fails, else pointer to the given land use of CLandUse class type.

This is the caller graph for this function:

CBMPData::FindPumpCurve (CString strPCurveID*)*

This function searches for the given pump curve in the PumpList.

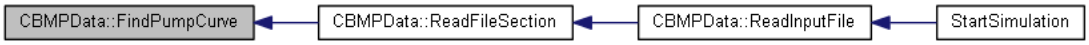
Parameter:

strPCurveID	Unique identifier for the pump object.

Returns:

Null if it fails, else pointer to the given pump of CPUMP class type.

This is the caller graph for this function:



CBMPData::LoadClimateTSData (COleDateTime startDate*, COleDateTime* endDate*)*

This function reads the climate time series data for the daily air temperature (maximum and minimum) and PET values.

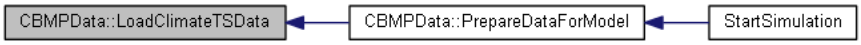
Parameters:

startDate	Start date of model simulation.
endDate	End date of model simulation.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:



CBMPData::OpenOutputFiles (const CString & runID*)*

This function opens the BMP output time series file.

Parameter:

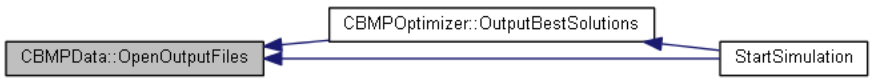
runID	The model simulation type (Init, PreDev, and PostDev).

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:

This is the caller graph for this function:



*CBMPData::OutputFileHeaderForTradeOffCurve (FILE * fp)*

This function writes the header information for the cost-effectiveness curve output file. The file contains the information about the best solution ID, the total cost of the project, the evaluation factor for the cost-effectiveness curve, and the optimized decision variables.

Parameter:

fp	The pointer to the cost-effectiveness curve output file.

This is the caller graph for this function:

CBMPData::PrepareDataForModel ()

This function processes the model input data to start the BMP simulation. It loads the flow and pollutant time series data from the external land use, point source, and the model output time series if needed. It also loads the weather data from the climate time series file. It processes the pollutant data and assigns the routing sequence to the BMP network.

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:

This is the caller graph for this function:

CBMPData::ProcessPollutantData ()

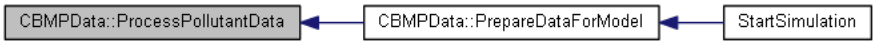
This function processes the pollutant data to be used in the model simulation. It splits the total sediment into three sediment classes (Sand, Silt, and Clay) if the sediment is defined as a total

sediment in the pollutant definition. This function resizes the data arrays for the pollutant 1st order decay rates, *K'*, *C**, and underdrain removal rates to incorporate the sand, silt, and clay if the total sediment is simulated.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:



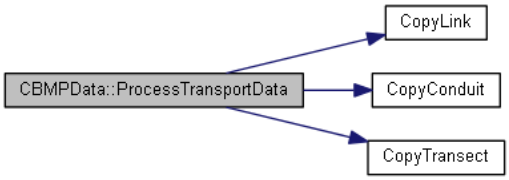
CBMPData::ProcessTransportData ()

This function processes the conveyance data to be used in the model simulation. It initializes the conduit-related parameters and creates a copy of the Link, Conduit, and Transect input data to be used for the optimization run.

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:



This is the caller graph for this function:

CBMPData::ReadBestPopFile (int nBestPopId*)*

This function reads the best solutions output file (BestSolutions.out) for the given solution. It reads the optimized decision variables for the given best solution to be used for the tier-two optimization run.

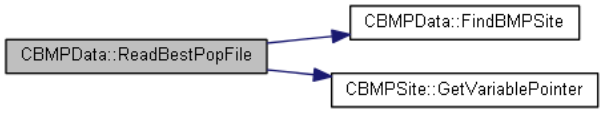
Parameter:

nBestPopId	The unique identifier for the best solution in the output file.

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:



This is the caller graph for this function:

*CBMPData::ReadDataLine (FILE * fp, CString & strData)*

This function reads a single data line from the model input text file.

Parameters:

fp	The pointer to the model input text file.
strData	The model data line from the input text file.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:

*CBMPData::ReadFileSection (FILE * fp, int nSection)*

This function reads the model input parameters for the selected card from a section of the input file. It calls several other companion functions to read and initialize the model input parameters.

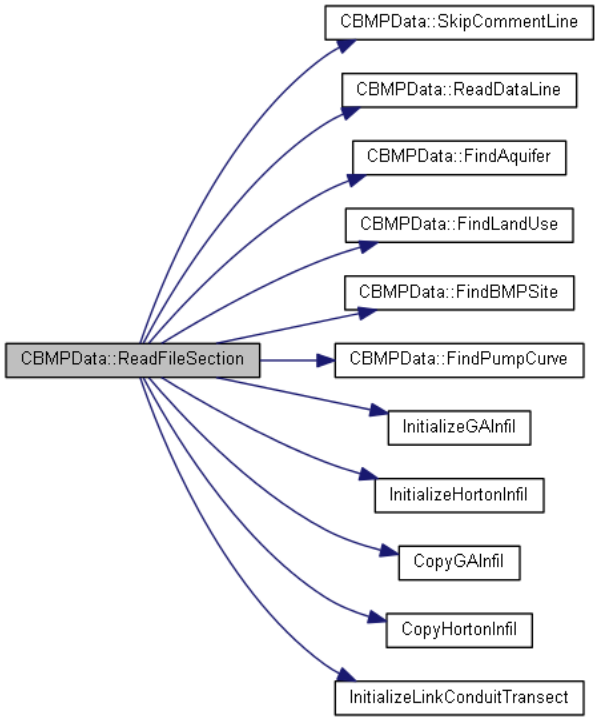
Parameters:

fp	The pointer to the model input text file.
nSection	The section of the model input text file (referred as card number).

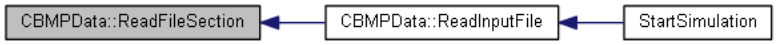
Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:



This is the caller graph for this function:



CBMPData::ReadInputFile (CString strFileName*)*

This function reads the model input text file. It parses the input text file into several sections and calls the CBMPData::ReadFileSection to read each section.

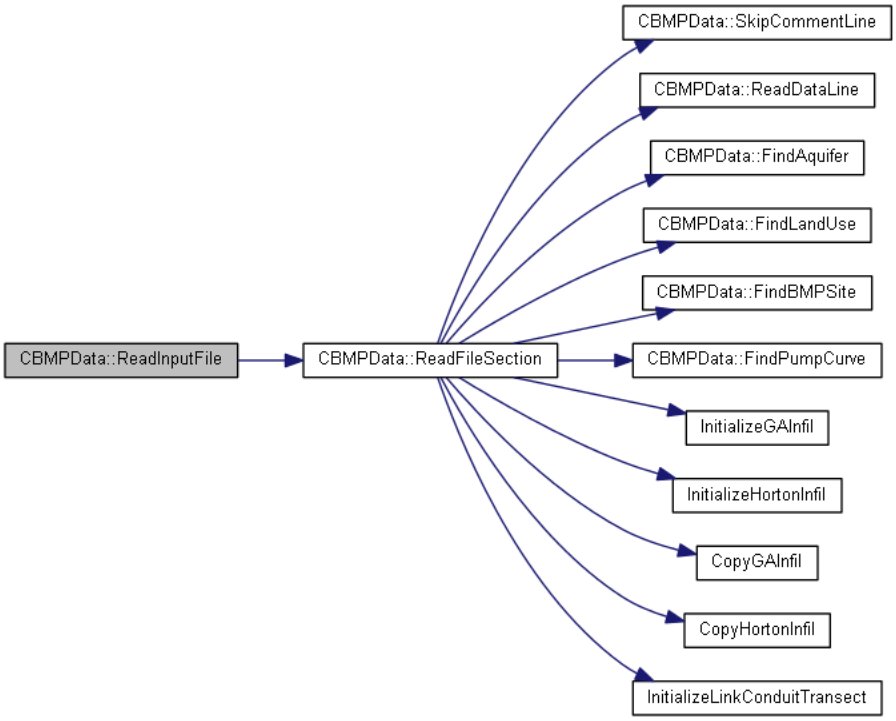
Parameter:

strFileName	The full path for the model input text file.

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:



This is the caller graph for this function:

CBMPData::RoutingCycleExist (CBMPSite * pBMPSite*)*

This function checks the closed loops in the BMP routing network. It checks the connected downstream BMPs of the selected BMP site for the closed loops.

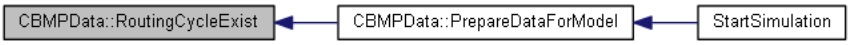
Parameter:

pBMPSite	The pointer to the given BMP site.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:



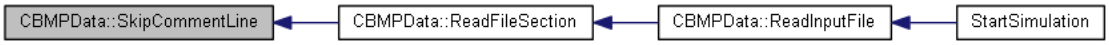
*CBMPData::SkipCommentLine (FILE * fp)*

This function skips the comment lines while reading the model input file.

Parameter:

fp	The pointer to the model input text file.

This is the caller graph for this function:



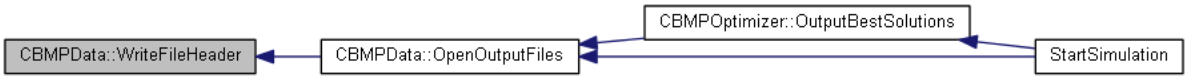
*CBMPData::WriteFileHeader (FILE * fp, int NWQ)*

This function writes the header information for all the output parameters in the time series output file.

Parameters:

fp	The pointer to the model time series output file.
NWQ	The number of modeled pollutants.

This is the caller graph for this function:



Member Data Documentation

CObList CBMPData::AquiferList

This is the list of aquifer objects (for external land simulation).

CObList CBMPData::bmpsiteList

This is the list of total BMP site objects.

COleDateTime CBMPData::endDate

This is the DATE data type for the model simulation end time.

double CBMPData::lfLatitude

This is the latitude in decimal degrees and is required if the ET flag is equal to 2.

double CBMPData::lfMaxRunTime

This is the user-specified maximum simulation run time.

double CBMPData::lfmonET[12]

This is the monthly PET rate (inch/day) if the ET flag is equal to 0; the monthly pan coefficient (multiplier to PET value) if the ET flag is equal to 1; the monthly variable coefficient to calculate PET values if the ET flag is equal to 2.

double CBMPData::lfStopDelta

This is the user-specified criteria for stopping the optimization iterations.

CObList CBMPData::luList

This is the list of total land use objects.

int CBMPData::m_nNum

This is the number of weather parameters in the climate file.

int CBMPData::m_nSedQualFlag

This is the flag to distinguish if there is any adsorbed pollutant.

BMPCOST * CBMPData::m_pBMPcost

This is an array of BMPCOST structure data type for the total BMPs in the project.

double * CBMPData::m_pDataClimate

This is an array for storing the climate time series data.

POLLUTANT * CBMPData::m_pPollutant

This is an array of POLLUTANT structure data type for the total pollutants in the project.

int CBMPData::nAdjVariable

This is the number of decision variables (BMP parameters that vary during the iteration process) in the project.

int CBMPData::nAREABMP

This is the number of disconnected impervious area BMP types in the project.

int CBMPData::nBIORETENTION

This is the number of bioretention BMP types in the project.

int CBMPData::nBMPA

This is the number of class-A BMP types in the project.

int CBMPData::nBMPB

This is the number of class-B BMP types in the project.

int CBMPData::nBMPC

This is the number of class-C BMP types in the project.

int CBMPData::nBMPD

This is the number of class-D BMP types in the project.

int CBMPData::nBMPE

This is the number of class-E BMP types in the project.

int CBMPData::nBMPTimeStep

This is the model timestep for the BMP simulation.

int CBMPData::nBMPtype

This is the total number of BMP types in the project.

int CBMPData::nBUFFERSTRIP

This is the number of buffer strip BMP types in the project.

int CBMPData::nCISTERN

This is the number of cistern BMP types in the project.

int CBMPData::nDRYPOND

This is the number of dry pond BMP types in the project.

int CBMPData::nETflag

This is the ET Flag (0-constant monthly PET, 1-daily PET from the time series, 2-calculate daily PET from the daily temperature data).

int CBMPData::nEvalFactor

This is the total number of evaluation factors.

int CBMPData::nGREENROOF

This is the number of green roof BMP types in the project.

int CBMPData::nINFILTRATIONTRENCH

This is the number of infiltration trench BMP types in the project.

int CBMPData::nLandSimulation

This is the land simulation option (0-external, 1-internal).

int CBMPData::nLANDTimeStep

This is the model timestep for the land simulation (minutes).

int CBMPData::nNWQ

This is the number of modeled pollutants (sediment is modeled as sand, silt, and clay).

int CBMPData::nOutputTimeStep

This is the model timestep for the BMP output time series.

int CBMPData::nPollutant

This is the number of user-defined pollutants.

int CBMPData::nPOROUSPAVEMENT

This is the number of porous pavement BMP types in the project.

int CBMPData::nRAINBARREL

This is the number of rain barrel BMP types in the project.

int CBMPData::nREGULATOR

This is the number of regulator BMP types in the project.

int CBMPData::nRunOption

This is the unique identifier for simulation run (0-no optimization, 1-minimize cost, 2-cost-effectiveness curve).

int * CBMPData::nSedflag

This is an array of pollutants storing the sediment flag (1-sand, 2-silt, 3-clay).

int CBMPData::nSolution

This is the number of best solutions time series to output.

int CBMPData::nStrategy

This is the flag for the optimization strategy (1-Scatter Search, 2-Genetic Algorithm).

int CBMPData::nSWALE

This is the number of swale BMP types in the project.

int CBMPData::nWETPOND

This is the number of wet pond BMP types in the project.

double * CBMPData::polmultiplier

This is an array of the total number of pollutants that stores the multiplier for each pollutant loading rate.

CObList ***CBMPData::PumpList***

This is the list of pump objects.

CObList ***CBMPData::routeList***

This is the list of BMP site objects sorted in the routing order from upstream to downstream.

CObList ***CBMPData::siteluList***

This is the list of land use objects for the given BMP site.

CObList ***CBMPData::sitepsList***

This is the list of point source objects for the given BMP site.

COleDateTime ***CBMPData::startDate***

This is the DATE data type for the model simulation start time.

CString ***CBMPData::strClimateFileName***

This is the full path for the climate time series file (required if PET is read from the climate time series or calculated in the model).

CString ***CBMPData::strError***

This is the string to save error messages.

CString ***CBMPData::strInputDir***

This is the land output directory (containing land output time series) that is input to BMP simulation module in *SUSTAIN*.

CString ***CBMPData::strMixLUFileName***

This is the full path of mixed land use output file (for internal land simulation).

CString ***CBMPData::strOutputDir***

This is the BMP simulation output directory where the BMP output time series are stored.

CString ***CBMPData::strPreLUFileName***

This is the full path of pre-developed land use output file (for internal land simulation).

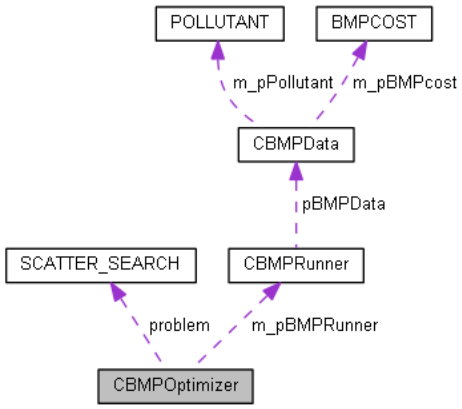
The documentation for this class was generated from the following files:

- BMPData.h
- BMPData.cpp

CBMPOptimizer Class Reference

#include "BMPOptimizer.h"

Collaboration diagram for CBMPOptimizer:



Public Member Functions

- CBMPOptimizer ()
- CBMPOptimizer (CBMPRunner *pBMPRunner)
- virtual ~CBMPOptimizer ()
- void InitRefSet ()
- void ResetRefSet ()
- void CombineRefSet ()
- void InitProblem (int nVar, int b1, int b2, int pSize, bool localSearch)
- void Combine_inc (double *x, double *y, double **offsprings, int number)
- void TryAddRefSet1 (double *sol)
- void TryAddRefSet2 (double *sol)
- void UpdateRefSet2 ()
- void TryAddEvaluation (double *sol, double current_value)
- void GetOrderIndices (int *indices, double *pesos, int num, int tipo)
- void PerformSearch ()
- void OutputFileHeader (CString header, FILE *fp)
- void OutputBestSolutions ()
- bool IsNewSolution (double **solutions, int dim, double *sol)
- double GenerateValue (int a)
- double Evaluate (double *sol)
- double Evaluate_MinCost (double *sol)
- double DistanceToRefSet1 (double *sol)
- double DistanceToRefSet (double *sol)

Public Attributes

- int nMaxIter
- int nRunCounter
- int nMaxRun
- int nRUN_BATCH
- double m_lfPrevResult
- double m_lfPrevValue
- double ** m_pVariables
- CBMPRunner * m_pBMPRunner
- FILE * m_pAllSolutions
- SCATTER_SEARCH problem

Detailed Description

This is the class that runs the BMP optimization module for minimizing the project cost to achieve an evaluation target. The module uses evolutionary Scatter Search optimization technique to perform the searches for optimal combinations of BMPs that meet the user-defined decision criteria. The Scatter Search technique is more efficient than NSGA-II because the search is more focused around a target.

Constructor & Destructor Documentation

CBMPOptimizer::CBMPOptimizer ()

This is the CBMPOptimizer class constructor (default).

*CBMPOptimizer::CBMPOptimizer (CBMPRunner * pBMPRunner)*

This is the CBMPOptimizer class constructor that initializes the class variables. It assigns a pointer to CBMPRunner class.

Parameters:

in,out	*pBMPRunner*	If non-null, a pointer to CBMPRunner class.

CBMPOptimizer::~CBMPOptimizer () [virtual]

This is the CBMPOptimizer class destructor.

Member Function Documentation

*void CBMPOptimizer::Combine_inc (double * x, double * y, double ** offsprings, int number)*

This function constructs new solution points from the reference solution points. The method consists of finding linear combinations of reference solutions. The number of solutions created from the linear combination of two reference solutions depends on the membership of the solutions being combined. These combinations are based on the following three types, assuming that the reference solutions are x' and x'':

C1: $x_1 = x' - d$

C2: $x_2 = x' + d$

C3: $x_3 = x'' + d$

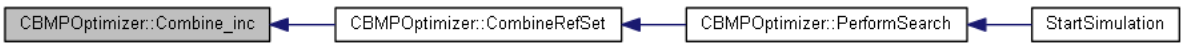
where

$d = r\,(x'' - x')/2$, r is a random number in the range $(0, 1)$, and x is the new solution point.

Parameters:

	number	The number of new solutions to be constructed
in,out	*x*	An array of reference solution set x (as x' in the equations above)
in,out	*y*	An array of reference solution set y (as x'' in the equations above)
in,out	*offsprings*	A two-dimensional array of new solutions

This is the caller graph for this function:



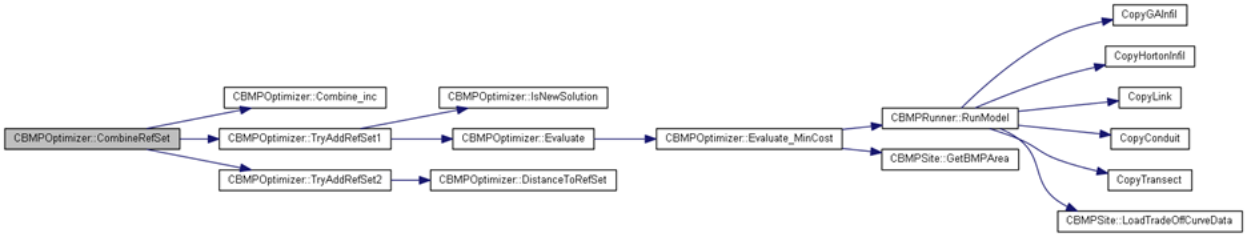
void CBMPOptimizer::CombineRefSet ()

This function generates new solutions by combining solutions from *RefSet1* and *RefSet2*. It calls function CBMPOptimizer::Combine_inc to perform the Combination Method. The following rules are used to generate solutions with these three types of linear combinations:

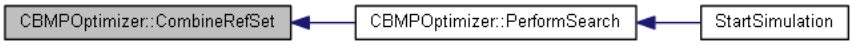
- If both x' and x'' are elements of *RefSet1*, generate four solutions by applying C1 and C3 once and C2 twice.

- If only one of *x'* and *x"* is a member of *RefSet1*, generate three solutions by applying C1, C2 and C3 once.
- If neither *x'* nor *x"* is a member of *RefSet1*, generate two solutions by applying C2 once and randomly choosing between applying C1 or C3.

This is the call graph for this function:



This is the caller graph for this function:



*double CBMPOptimizer::DistanceToRefSet (double * sol)*

This function calculates the Euclidean distance of a solution to the solutions in both reference set 1 (*RefSet1*) and reference set 2 (*RefSet2*).

Parameter:

in,out	*sol*	An array of solution

Returns:

The minimum distance between a solution and all the solutions in both reference sets.

This is the caller graph for this function:

*double CBMPOptimizer::DistanceToRefSet1 (double * sol)*

This function calculates the Euclidean distance of a solution to the solutions in reference set 1 (*RefSet1*).

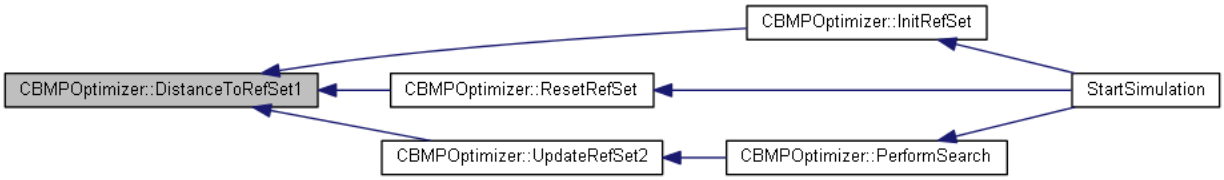
Parameter:

in,out	*sol*	An array of solution

Returns:

The minimum distance between a solution and the solutions in reference set 1 (RefSet1).

This is the caller graph for this function:



*double CBMPOptimizer::Evaluate (double * sol)*

This function calls CBMPOptimizer::Evaluate_MinCost function.

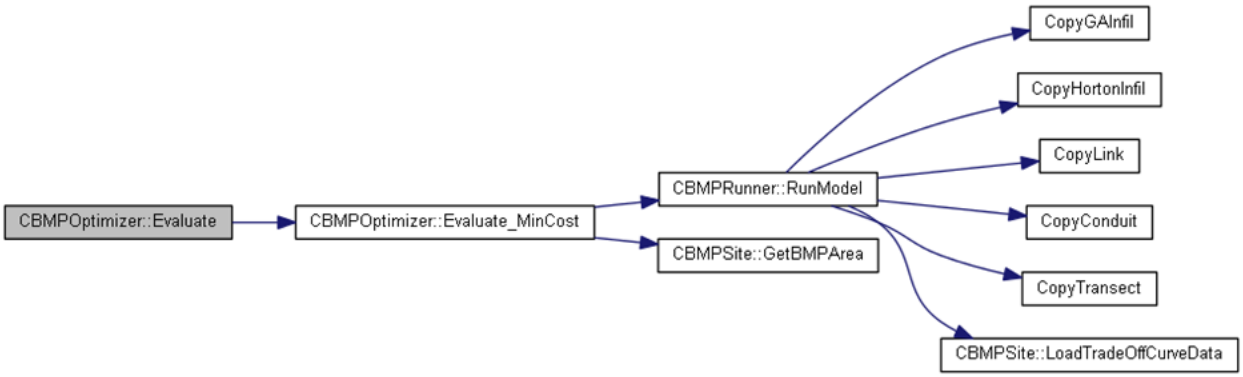
Parameter:

in,out	*sol*	An array of solution

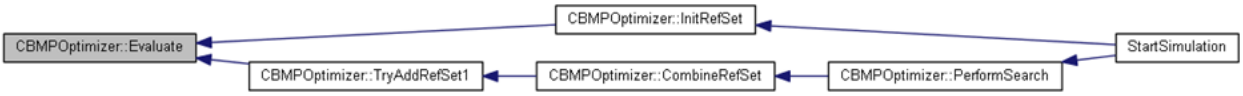
Returns:

The value of the solution evaluated. The value is a composite measurement of the solution's merit, considering both its performance and cost.

This is the call graph for this function:



This is the caller graph for this function:



*double CBMPOptimizer::Evaluate_MinCost (double * sol)*

This function returns the value of a solution. The value is calculated on the basis of a solution's cost and performance. The performance of a solution is measured by whether the target is met. If the target is not met, measure how close it is to the target.

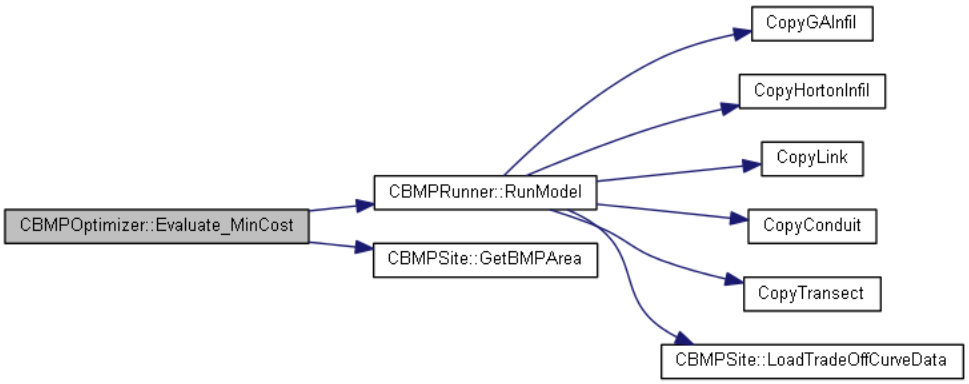
Parameter:

in,out	*sol*	An array of solution

Returns:

The value of the solution evaluated.

This is the call graph for this function:



This is the caller graph for this function:

double ***CBMPOptimizer::GenerateValue*** *(int* **a***)*

This function generates a number in the specified low and high bounds of a decision variable.

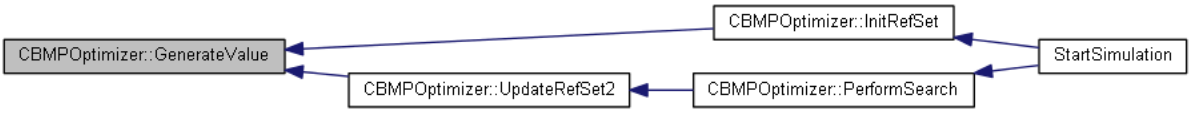
Parameter:

a	Decision variable index

Returns:

A number for decision variable between its low and high bounds.

This is the caller graph for this function:



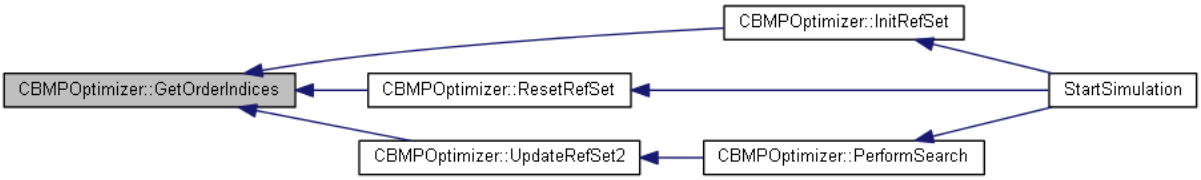
void ***CBMPOptimizer::GetOrderIndices*** *(int ** **indices***, double ** **pesos***, int* **num***, int* **tipo***)*

This function sorts solutions in a set.

Parameters:

	num	Number of solutions to be sorted
	tipo	Sorting order; 1 indicates increasing, -1 indicates decreasing
in,out	*indices*	An array of indexes
in,out	*pesos*	An array of values of the solutions

This is the caller graph for this function:



void ***CBMPOptimizer::InitProblem*** *(int* nVar*, int* b1*, int* b2*, int* pSize*, bool* localSearch*)*

This function initializes all the arrays, including the problem solution set, reference set 1, and reference set 2.

Parameters:

nVar	Number of decision variables
b1	Number of solutions in *RefSet1*
b2	Number of solutions in *RefSet2*
pSize	Population size
localSearch	Flag for local search, it is intended for continuous variable, not applicable in *SUSTAIN*.

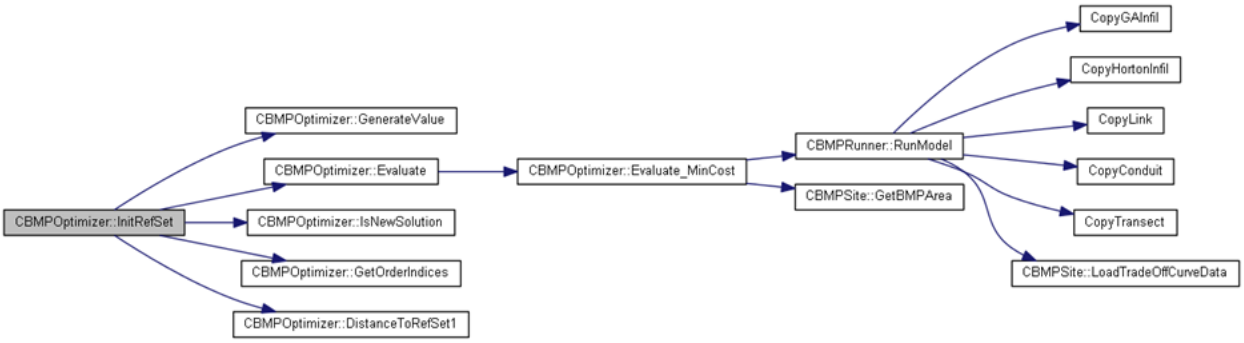
This is the call graph for this function:

This is the caller graph for this function:

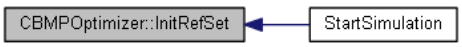
void ***CBMPOptimizer::InitRefSet*** *()*

This function populates the initial problem solution set, chooses the best *b1* solutions to populate reference set 1, and chooses *b2* solutions with the maximum distance to the reference set 1 solutions to populate reference set 2.

This is the call graph for this function:



This is the caller graph for this function:



bool ***CBMPOptimizer::IsNewSolution*** *(double *** solutions*, int* dim*, double ** sol*)*

This function checks whether a newly generated solution is identical to the solutions in the reference set.

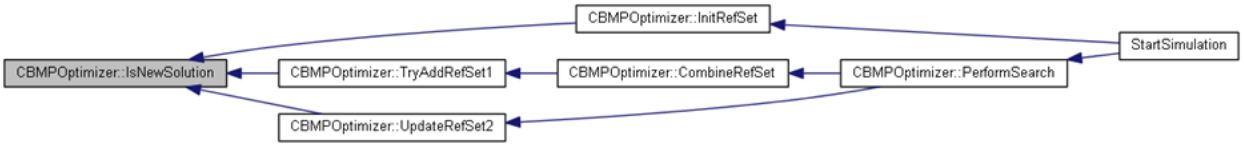
Parameters:

	dim	Dimensions of the solution array
in,out	*sol*	An array of solution
in,out	*solutions*	A set of solutions (two dimentional array)

Returns:

True if the solution is different from any solution in the reference set, false if the solution is identical to a solution in the reference set.

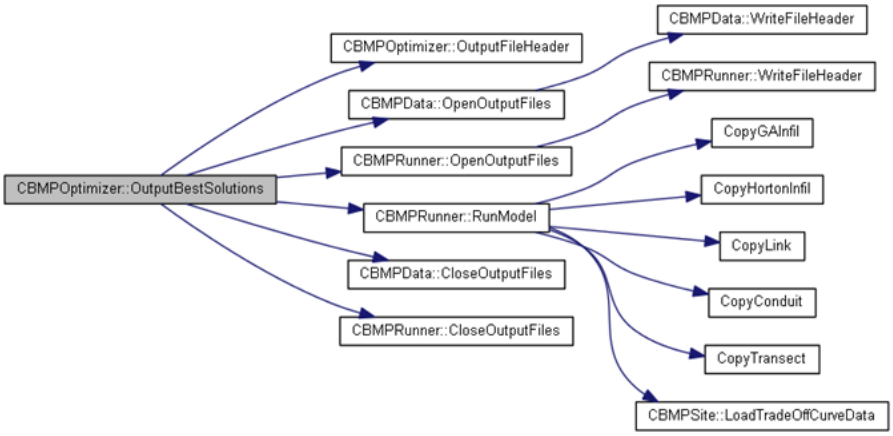
This is the caller graph for this function:



void CBMPOptimizer::OutputBestSolutions ()

This function writes the best solutions to the BestSolutions.out output file.

This is the call graph for this function:



This is the caller graph for this function:

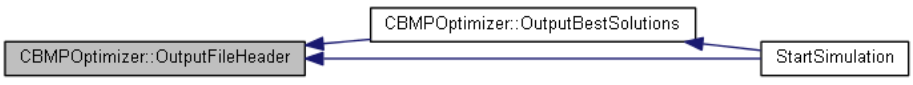
void CBMPOptimizer::OutputFileHeader (CString **header**, *FILE ** **fp**)

This function writes the output file header.

Parameters:

	header	Header information to be written in the output file
in,out	*fp*	Pointer to the output file

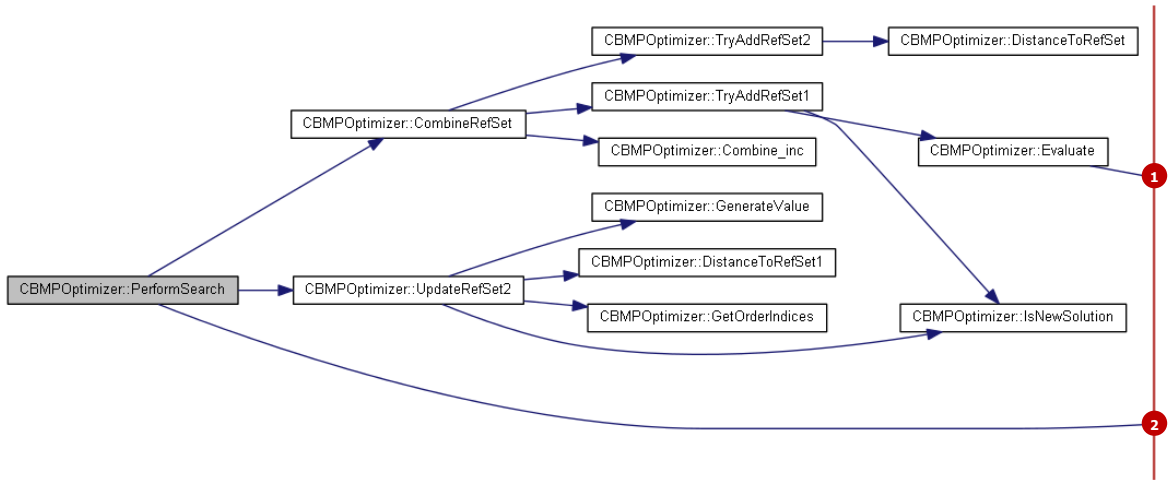
This is the caller graph for this function:

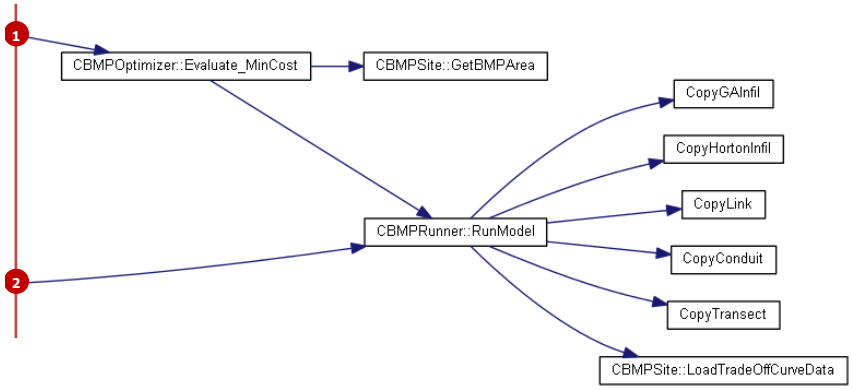


void CBMPOptimizer::PerformSearch ()

This function performs the search process after initializing the reference sets. It combines the reference sets, updates reference set, and if the stopping criteria are met, it stops the search.

This is the call graph for this function:



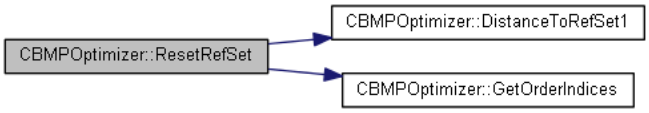


This is the caller graph for this function:

void CBMPOptimizer::ResetRefSet ()

This function is not used in the current version of *SUSTAIN*. It was to reset the reference set, if the Scatter Search was used for the Cost-Effectiveness Curve option.

This is the call graph for this function:



This is the caller graph for this function:

*void CBMPOptimizer::TryAddEvaluation (double * sol, double current_value)*

This function is not used in the current version of *SUSTAIN*. It was to be used if the Scatter Search was used for the Cost-Effectiveness Curve option.

Parameters:

	current_value	Current value of the evaluation factor
in,out	*sol*	An array of solution

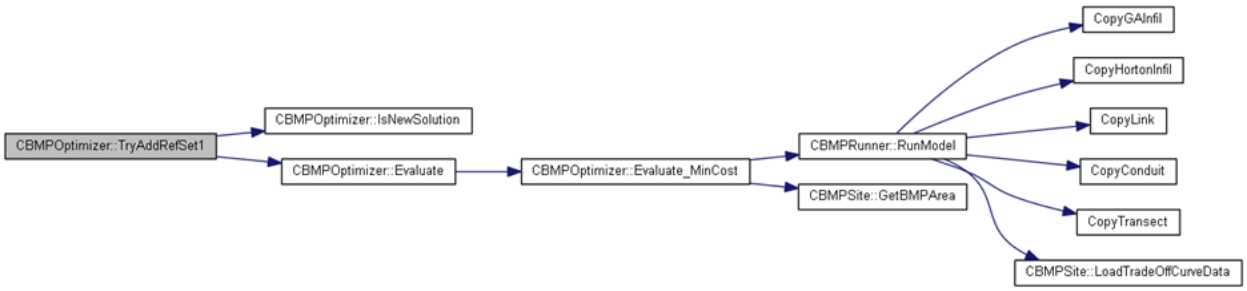
*void CBMPOptimizer::TryAddRefSet1 (double * sol)*

In the process of combining reference sets, this function evaluates the newly combined solutions against the existing solutions in reference set 1. If the new solutions are superior, they will replace the inferior solutions currently in reference set 1.

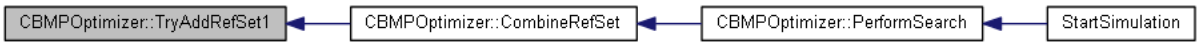
Parameter:

in,out	*sol*	An array of solution to be evaluated

This is the call graph for this function:



This is the caller graph for this function:



*void CBMPOptimizer::TryAddRefSet2 (double * sol)*

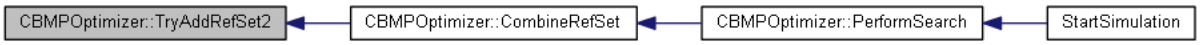
In the process of combining reference sets, this function evaluates the newly combined solutions against the existing solutions in reference set 2. If the new solutions are superior, they will replace the inferior solutions in reference set 2.

Parameter:

in,out	*sol*	An array of solution to be evaluated

This is the call graph for this function:

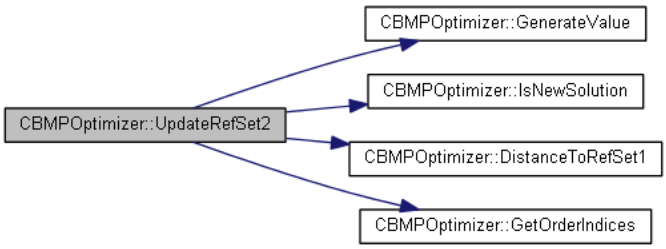
This is the caller graph for this function:



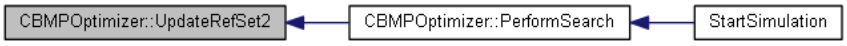
void CBMPOptimizer::UpdateRefSet2 ()

This function updates reference set 2 by replacing the old solution with newly generated solutions to increase diversity.

This is the call graph for this function:



This is the caller graph for this function:



Member Data Documentation

double CBMPOptimizer::m_lfPrevResult

This is the cost of the previously evaluated solution.

double CBMPOptimizer::m_lfPrevValue

This is the value of the previously evaluated solution.

FILE * CBMPOptimizer::m_pAllSolutions

This is a pointer to the AllSolutions.out output file.

CBMPRunner * CBMPOptimizer::m_pBMPRunner

This is a pointer to CBMPRunner class.

double ** CBMPOptimizer::m_pVariables

This is the two-dimensional array that stores the decision variables for all solutions.

int CBMPOptimizer::nMaxIter

This is the maximum number of iterations in an optimization run.

int CBMPOptimizer::nMaxRun

This is the maximum number of runs.

int CBMPOptimizer::nRUN_BATCH

This is an indicator of batch mode run; if it is equal to 1, it skips all messages and time series output files.

int CBMPOptimizer::nRunCounter

This is the counter for the number of optimization runs.

SCATTER_SEARCH CBMPOptimizer::problem

This is a data structure class of SCATTER_SEARCH type.

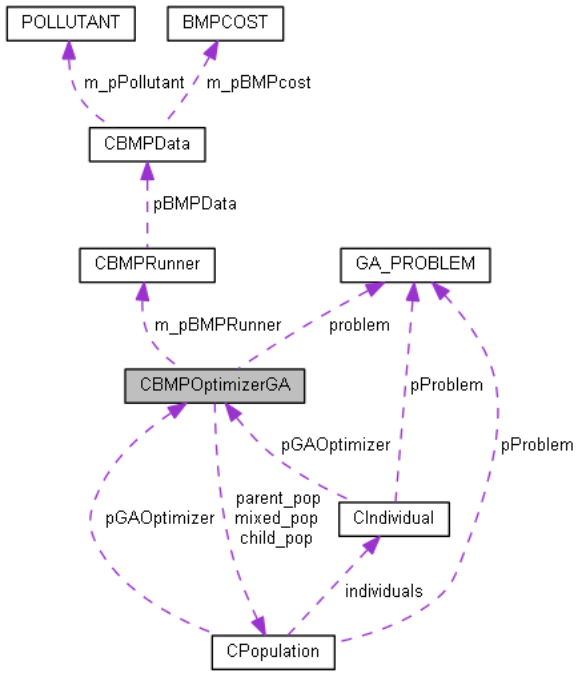
The documentation for this class was generated from the following files:

- BMPOptimizer.h
- BMPOptimizer.cpp

CBMPOptimizerGA Class Reference

#include "BMPOptimizerGA.h"

Collaboration diagram for CBMPOptimizerGA:



Public Member Functions

- CBMPOptimizerGA ()
- CBMPOptimizerGA (CBMPRunner *pBMPRunner)
- virtual ~CBMPOptimizerGA ()
- bool OpenOutputFiles ()
- bool CloseOutputFiles ()
- bool LoadData ()
- bool ValidateParams ()
- bool InitProblem ()
- bool PerformSearch ()
- bool EvaluateSolution (double *xreal, double *obj, double *bmpcost, double *constr)
- bool Evaluate_TradeOff (double *xreal, double *obj, double *bmpcost, double *constr)
- void OutputBestPopulation ()
- void Selection (CPopulation *old_pop, CPopulation *new_pop)
- void CrossOver (CIndividual *parent1, CIndividual *parent2, CIndividual *child1, CIndividual *child2)
- void CrossOverReal (CIndividual *parent1, CIndividual *parent2, CIndividual *child1, CIndividual *child2)
- void Merge (CPopulation *pop1, CPopulation *pop2, CPopulation *pop3)
- void FillNondominatedSort (CPopulation *mixed_pop, CPopulation *new_pop)
- void CrowdingFill (CPopulation *mixed_pop, CPopulation *new_pop, int count, int front_size, void *list)
- void OutputFileHeader (CString header, FILE *fp)
- CIndividual * Tournament (CIndividual *ind1, CIndividual *ind2)

Public Attributes

- int nRunCounter

- int nMaxRun
- int nRUN_BATCH
- double ** m_pVariables
- GA_PROBLEM problem
- CPopulation * parent_pop
- CPopulation * child_pop
- CPopulation * mixed_pop
- CBMPRunner * m_pBMPRunner
- FILE * fpBestPop
- FILE * m_pAllSolutions

Detailed Description

This class defines the functions that are necessary for carrying out the BMP optimization process using the NSGA-II algorithm. The functions include construction and destruction of the CBMPOptimizerGA class, opening and closing the optimization output files, loading optimization parameters, validating the optimization parameters, initializing the optimization problem, and performing the optimization search.

Constructor & Destructor Documentation

CBMPOptimizerGA::CBMPOptimizerGA ()

This is the CBMPOptimizerGA class constructor (default).

*CBMPOptimizerGA::CBMPOptimizerGA (CBMPRunner * pBMPRunner)*

This is the CBMPOptimizerGA class constructor that initializes the class variables. It assigns a pointer to the CBMPRunner class.

Parameter:

in,out	*pBMPRunner*	If non-null, a pointer to CBMPRunner class.

CBMPOptimizerGA::~CBMPOptimizerGA () [virtual]

This is the CBMPOptimizerGA class destructor.

This is the call graph for this function:

Member Function Documentation

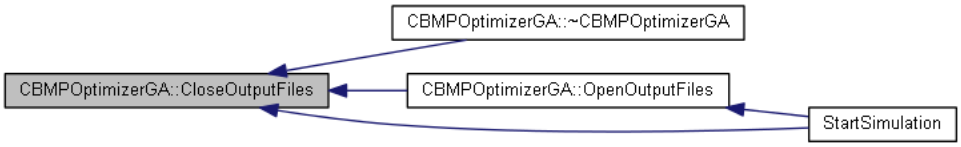
bool CBMPOptimizerGA::CloseOutputFiles ()

This function closes the output files associated with the optimization process. The output files are the AllSolutions.out and the BestPop.out.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:



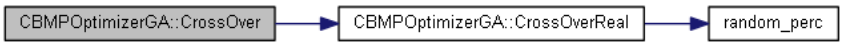
void ***CBMPOptimizerGA::CrossOver*** *(**CIndividual** * parent1, **CIndividual** * parent2, **CIndividual** * child1, **CIndividual** * child2)*

This function performs simulated binary crossover (SBX) for two individuals in a population. It calls the CBMPOptimizerGA::CloseOutputFiles function.

Parameters:

in,out	*parent1*	A pointer to Individual #1 in the current population.
in,out	*parent2*	A pointer to Individual #2 in the current population.
in,out	*child1*	A pointer to new Individual #1 after crossover.
in,out	*child2*	A pointer to new Individual #2 after crossover.

This is the call graph for this function:



This is the caller graph for this function:

void ***CBMPOptimizerGA::CrossOverReal*** *(**CIndividual** * parent1, **CIndividual** * parent2, **CIndividual** * child1, **CIndividual** * child2)*

This function carries out the crossover for real variables. In this function two individuals are randomly selected in the current population, and two new individuals are created as a result of the crossover process. The SBX process is carried out only when the system-generated random percentage is smaller than the user-specified random percentage, the system-generated random percentage is smaller than 0.5, and the two individuals are different from each other.

Parameters:

in,out	*parent1*	A pointer to Individual #1 in the current population.
in,out	*parent2*	A pointer to Individual #2 in the current population.
in,out	*child1*	A pointer to new Individual #1 after crossover.
in,out	*child2*	A pointer to new Individual #2 after crossover.

This is the call graph for this function:

This is the caller graph for this function:

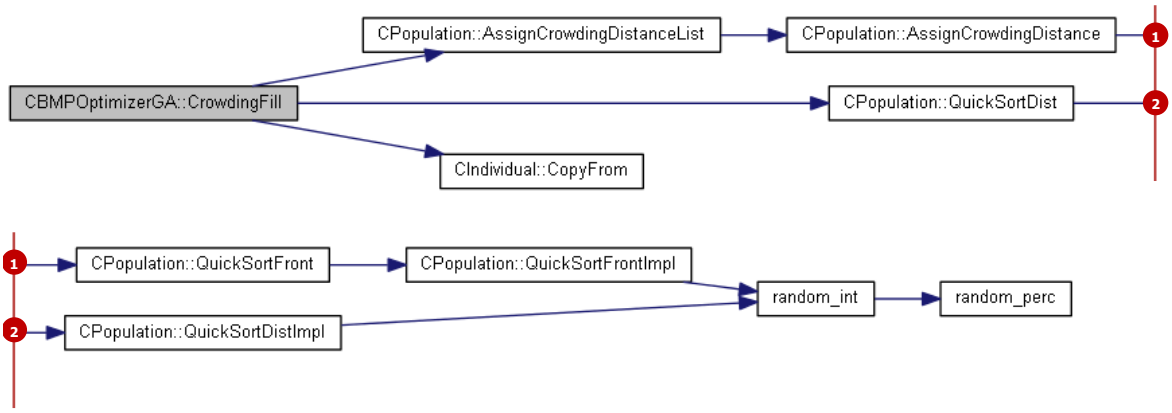
void ***CBMPOptimizerGA::CrowdingFill*** *(**CPopulation** * mixed_pop, **CPopulation** * new_pop, *int* count, *int* front_size, *void* * list)*

This function fills a population with individuals in the decreasing order of crowding distance. When carrying out the filling process, a crowding distance is first calculated for a mixed population and then the mixed population is sorted on the crowding distance of the individual solutions. The new population is filled with solutions from the mixed population.

Parameters:

	count	An index of the individual solution in the current front.
	front_size	The size of the current front.
in,out	*list*	A list that stores individual solutions in the current front.
in,out	*mixed_pop*	An array that stores the mixed population.
in,out	*new_pop*	An array that stores the new population.

This is the call graph for this function:



This is the caller graph for this function:

bool ***CBMPOptimizerGA::Evaluate_TradeOff*** *(double * xreal, double * obj, double * bmpcost, double * constr)*

This function carries out the BMP optimization process for cost-effectiveness curve generation. For individual solutions through the optimization process, it calculates the total BMP cost, total BMP surface area, total BMP excavation volume, total BMP surface storage volume, total BMP soil storage volume, and the total BMP underdrain storage volume. All solutions evaluated during the optimization process are exported to an output file.

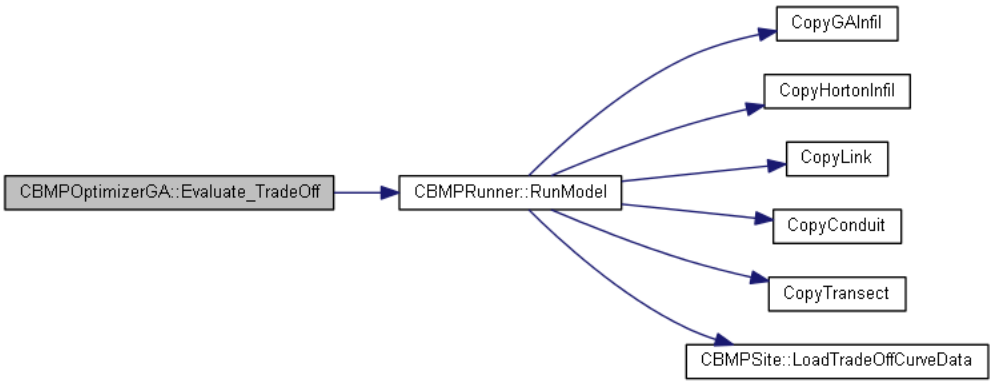
Parameters:

in,out	*xreal*	An array of real number decision variables.
in,out	*obj*	An array of the optimization targets.
in,out	*bmpcost*	An array of the costs of BMPs.
in,out	*constr*	An array of the constraints.

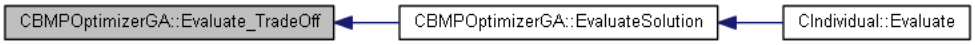
Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:



This is the caller graph for this function:



bool ***CBMPOptimizerGA::EvaluateSolution*** *(double* * **xreal**, *double* * **obj**, *double* * **bmpcost**, *double* * **constr**)

This function calls corresponding functions for carrying out the optimization process. The ArcGIS interface of *SUSTAIN* version 1.2 supports only the cost-effectiveness curve option.

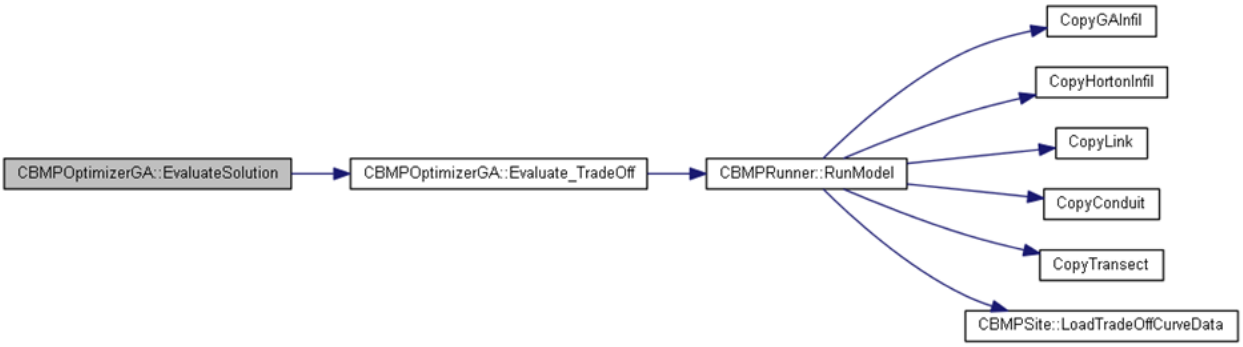
Parameters:

in,out	*xreal*	An array of real number decision variables.
in,out	*obj*	An array of the optimization targets.
in,out	*bmpcost*	An array of the costs of BMPs.
in,out	*constr*	An array of the constraints.

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:



This is the caller graph for this function:

void ***CBMPOptimizerGA::FillNondominatedSort*** *(**CPopulation** * **mixed_pop**, **CPopulation** * **new_pop**)*

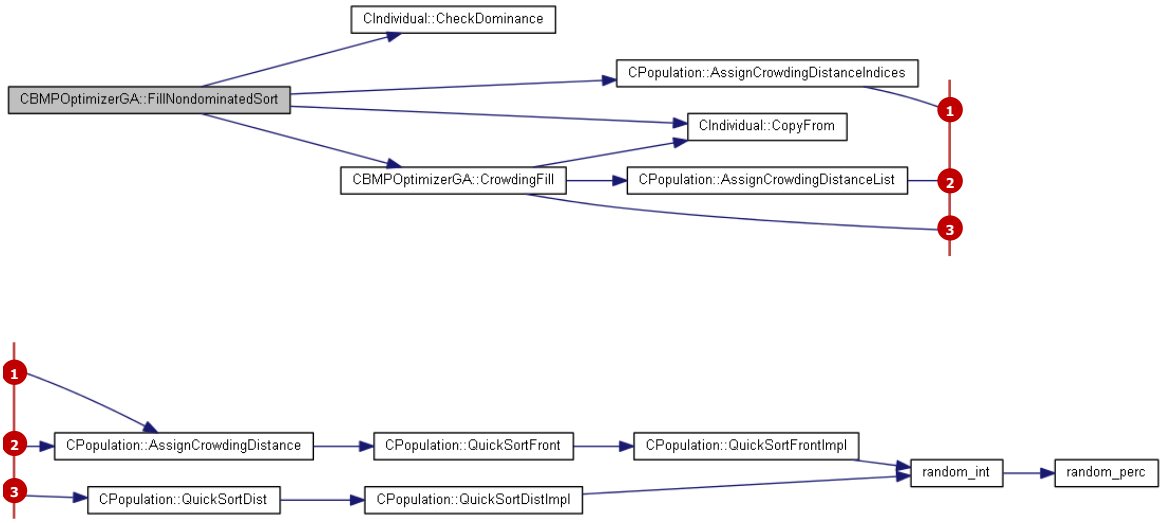
This function performs the non-dominated sorting of a mixed population and retrieves a new population after the sorting process is completed. During the sorting process, this function calls the CIndividual::CheckDominance function to check whether one solution dominate another. When filling the new population with individuals from the mixed population, it uses the

CIndividual::CopyFrom function. The CPopulation::AssignCrowdingDistanceIndices and the CBMPOptimizerGA::CrowdingFill functions are also used for filling the new population.

Parameters:

in,out	*mixed_pop*	An array that stores the mixed population.
in,out	*new_pop*	An array that stores the new population.

This is the call graph for this function:



This is the caller graph for this function:

bool CBMPOptimizerGA::InitProblem ()

This function initializes the optimization problem.

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:

This is the caller graph for this function:

bool CBMPOptimizerGA::LoadData ()

This function loads data that are used during the optimization process.

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:

This is the caller graph for this function:

*void CBMPOptimizerGA::Merge (CPopulation * pop1, CPopulation * pop2, CPopulation * pop3)*

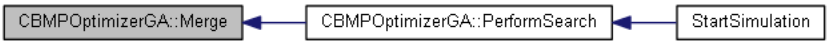
This function merges the parent population and the child population to create a mixed population.

Parameters:

in,out	*pop1*	An array that stores the parent population 1.
in,out	*pop2*	An array that stores the parent population 2.
in,out	*pop3*	An array that stores the mixed population 3.

This is the call graph for this function:

This is the caller graph for this function:



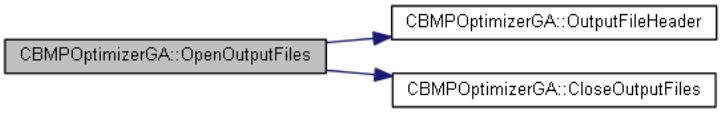
bool CBMPOptimizerGA::OpenOutputFiles ()

This function opens up the output files for writing. The output files being opened are the AllSolutions.out and BestSolutions.out. An error message is displayed if an output file cannot be opened.

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:



This is the caller graph for this function:

void CBMPOptimizerGA::OutputBestPopulation ()

This function exports the best population from the optimization process. The best population consists of individual solutions, each of which includes an array of BMPs. For each individual solution in the best population, the exported information includes Total Surface Area (*totalSurfaceArea*), Total Excavation Volume (*totalExcavatnVol*), Total Surface Storage Volume (*totalSurfStorVol*), Total Soil Storage Volume (*totalSoilStoreVol*), Total Underdrain Storage Volume (*totalUdrnStorVol*), total cost, and the value of each optimization target.

During the exporting process, individual solutions in the best population are first mapped to corresponding BMP design parameters. The BMP design parameters in each individual solution are then used for calculating the surface area, excavation volume, surface storage volume, soil storage volume, and underdrain storage volume for the individual solution. The aggregated values for each

of these parameters are calculated by summing corresponding values for all the BMPs in each individual, along with the total cost and the total value of each optimization target.

This is the caller graph for this function:

void ***CBMPOptimizerGA::OutputFileHeader*** *(CString* **header***, FILE ** **fp***)*

This function writes the header information to the specified output file.

Parameters:

	header	A string that contains the header information.
in,out	*fp*	A pointer to the output file

This is the caller graph for this function:

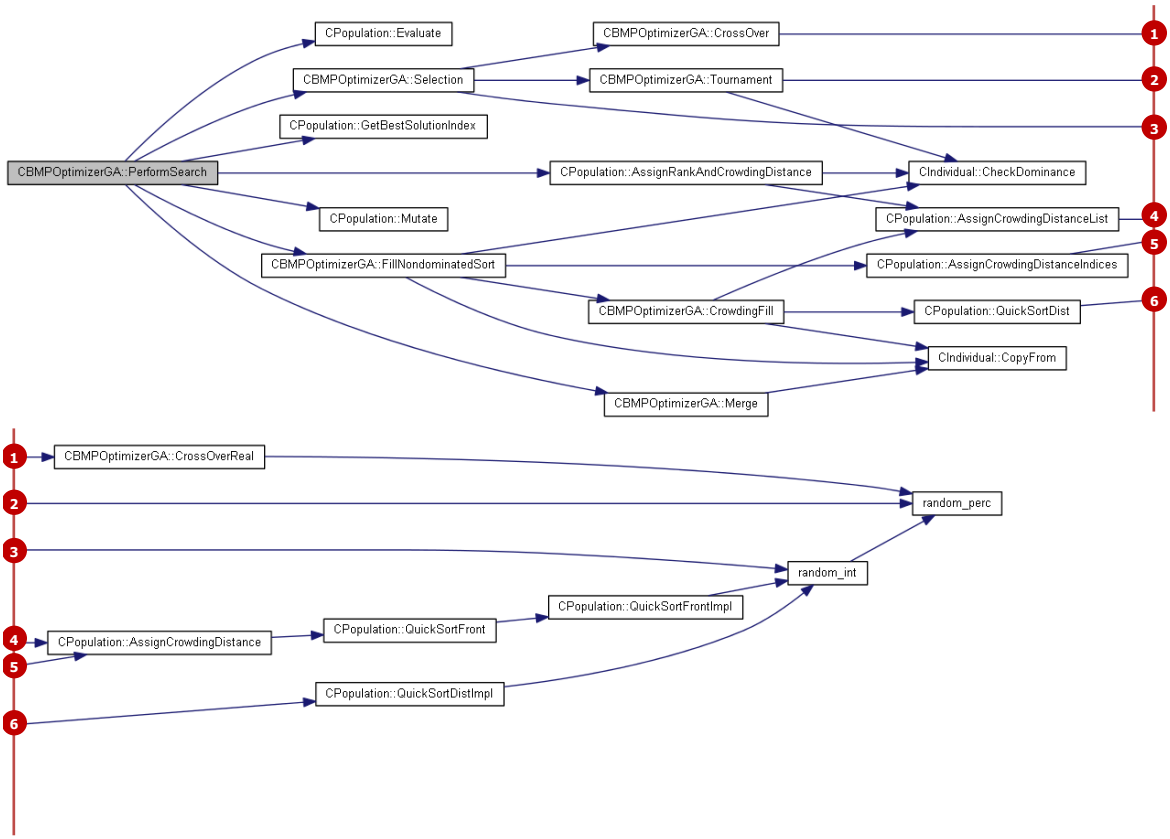
bool ***CBMPOptimizerGA::PerformSearch*** *()*

This function performs the optimization analysis. In this function, a parent population generated during the initialization process (CBMPOptimizerGA::InitProblem) is first evaluated, with the rank and the crowding distances of the individual solutions in the population assigned (CPopulation::AssignRankAndCrowdingDistance). The best solution in the population is also identified (CPopulation::GetBestSolutionIndex) along with its corresponding cost. Starting with the second generation, the optimization process first carries out a tournament selection (CBMPOptimizerGA::Selection; CBMPOptimizerGA::Tournament) to individuals in the parent population and thus creates a child population. A simulated binary crossover (CBMPOptimizerGA::CrossOver) is then carried out on individuals in the child population, to which the polynomial mutation (CPopulation::Mutate) is then applied. Each individual solution in the child population is then evaluated (CPopulation::Evaluate) according to the type of optimization (i.e. Cost-Effectiveness Curve Generation) (CBMPOptimizerGA::EvaluateSolution). After individual solutions in the child population are evaluated, the parent population and the child population are merged (CBMPOptimizerGA::Merge) to create a mixed population. A non-dominated sorting (CBMPOptimizerGA::FillNondominatedSort) is then carried out on the mixed population, creating a new parent population. Explicit elitism is maintained during the creation of this new parent population. The best solution in this new parent population is identified (CPopulation::GetBestSolutionIndex) along with its corresponding cost. The value of this cost is compared with the previous best solution total cost. If the difference between the two values is smaller than the user-specified value (lfStopDelta), a pop-up window asks the user to specify whether to continue the optimization process. The optimization process continues until the maximum number of generations is reached.

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:



This is the caller graph for this function:

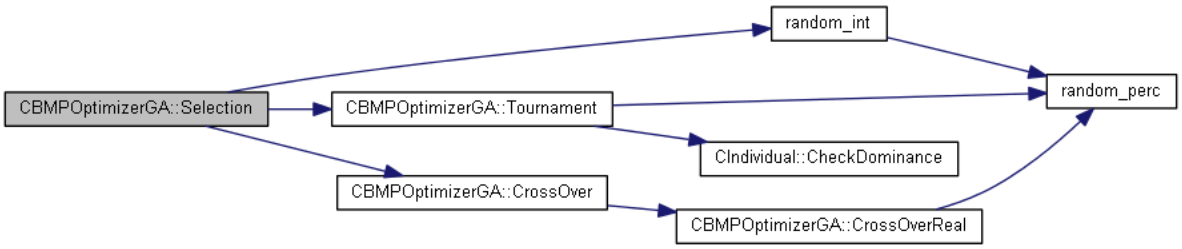
*void CBMPOptimizerGA::Selection (CPopulation * old_pop, CPopulation * new_pop)*

This function performs the tournament selection and crossover to an old population and creates a new population. The function randomly selects two pairs of individuals from the old population, and then calls the CBMPOptimizerGA::Tournament to carry out tournament selection on each pair. A crossover on the resulting pair of individual is then carried out through CBMPOptimizerGA::CrossOver. The crossover process results in two new individuals. The processes of tournament and crossover are then repeated for another four randomly selected individuals in the old population. This function is carried out until the population size is reached.

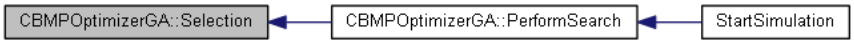
Parameters:

in,out	*old_pop*	An array that stores the old population.
in,out	*new_pop*	An array that stores the new population.

This is the call graph for this function:



This is the caller graph for this function:



*CIndividual * CBMPOptimizerGA::Tournament (CIndividual * ind1, CIndividual * ind2)*

This function carries out the tournament selection to two individual solutions. The tournament selection process chooses the non-dominant solution, and when the two solutions do not dominate each other, it chooses the solution with the larger crowding distance.

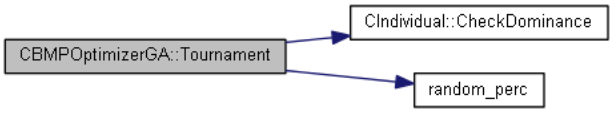
Parameters:

in,out	*ind1*	The first individual that participates in the tournament selection.
in,out	*ind2*	The second individual that participates in the tournament selection.

Returns:

The individual solution either non-dominant or the solution with the larger crowding distance.

This is the call graph for this function:



This is the caller graph for this function:

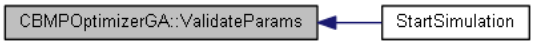
bool CBMPOptimizerGA::ValidateParams ()

This function checks the validity of optimization parameters. The check runs through the population size, number of generations, number of optimization objectives, number of constraints, real number variables, probability of real number crossover and mutation, and the distribution indexes for crossover and mutation.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:



Member Data Documentation

CPopulation * CBMPOptimizerGA::child_pop

This is an array of the child population of CPopulation class type.

FILE * CBMPOptimizerGA::fpBestPop

This is a pointer to the output file that stores the best population.

FILE * CBMPOptimizerGA::m_pAllSolutions

This is a pointer to the output file that stores all the solutions.

CBMPRunner * CBMPOptimizerGA::m_pBMPRunner

This is a pointer to the CBMPRunner class.

double ** CBMPOptimizerGA::m_pVariables

This is an array that stores the decision variables.

CPopulation * CBMPOptimizerGA::mixed_pop

This is an array that stores the mixed population of CPopulation class type.

int CBMPOptimizerGA::nMaxRun

This is the number of maximum runs for evaluation.

int CBMPOptimizerGA::nRUN_BATCH

This is an indicator of batch mode run. If it is equal to 1, it skips all messages and time series output files.

int CBMPOptimizerGA::nRunCounter

This is the counter for number of optimization runs.

CPopulation * CBMPOptimizerGA::parent_pop

This is an array that stores the parent population of CPopulation class type.

GA_PROBLEM CBMPOptimizerGA::problem

This is a data structure that defines the NSGA-II optimization parameters of GA_PROBLEM class type.

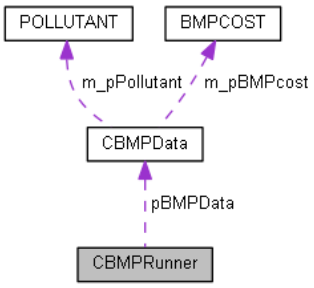
The documentation for this class was generated from the following files:

- BMPOptimizerGA.h
- BMPOptimizerGA.cpp

CBMPRunner Class Reference

#include "BMPRunner.h"

Collaboration diagram for CBMPRunner:



Public Member Functions

- CBMPRunner ()
- CBMPRunner (CBMPData *bmpData)
- virtual ~CBMPRunner ()
- void advect (double imat, double svol, double sro, double evol, double ero, double delts, double crrat, double &conc, double &romat)
- void bmp_a (int nInfiltMethod, int nInfil_Index, bool underdrain_on, int LANDtimestep, int BMPtimestep, int npeople, int ddays, int releasetype, int weirtype, int &counter, double oinflow, double BMParea, double orifice_dia, double orificeheight, double orificecoef, double weirwidth, double weirheight, double weirangle, double cisternoutflow, double soildepth, double soilporosity, double finalf, double vegparma, double holtpar, double udfinalf, double udsoildepth, double udsoilporosity, double FC, double WP, double ETrate, double &AET, double &perc, double &ovolume, double &ostage, double &infilt, double &orifice, double &weir, double &osa, double &ostorage, double &udout, double &seepage, double &pump, BMP_A *pBMP)
- void bmp_b (int nInfiltMethod, int nInfil_Index, bool underdrain_on, int LANDtimestep, int BMPtimestep, double oinflow, double BMPdepth, double BMPwidth, double BMPlength, double slope1, double slope2, double slope3, double man_n, double soildepth, double soilporosity, double finalf, double vegparma, double holtpar, double udfinalf, double udsoildepth, double udsoilporosity, double FC, double WP, double ETrate, double &AET, double &perc, double &ovolume, double &ostage, double &infilt, double &channel, double &weir, double &osa, double &ostorage, double &udout, double &seepage)
- void bmp_d (int nInfiltMethod, int nInfil_Index, int LANDtimestep, int BMPtimestep, double oinflow, double BMParea, double soildepth, double soilporosity, double finalf, double vegparma, double holtpar, double FC, double WP, double ETrate, double &AET, double &ovolume, double &ostage, double &infilt, double &weir, double &osa, double &seepage, CBMPSite *pBMPSite)
- void UpdateXareaStageSarea (double nvolume, double vol_max, double s_area_max, double BMPdepth, double BMPwidth, double BMPlength, double slope1, double slope2, double &x_area, double &nstage, double &sur_area)
- void RunModel (int nRunMode)
- bool OpenOutputFiles (const CString &runID, int nRunOption, int nRunMode)
- bool CloseOutputFiles ()
- void WriteFileHeader (int nRunOption, int nRunMode)

Public Attributes

- int optcounter
- int outcounter
- double lInitRunTime
- long nMaxRun
- CBMPData * pBMPData
- CProgressWnd * pWndProgress

- COleDateTime time_i
- FILE * fp
- int nRUN_BATCH

Detailed Description

This is the class that runs the BMP simulation module and computes the evaluation factors at the assessment points. It runs the baseline scenario without BMPs (Post-Dev) and with BMPs (Init) and a pristine condition (PreDev). The BMP optimization module calls this class to run the BMP simulation module for number of iteration to find the optimal solution.

Constructor & Destructor Documentation

CBMPRunner::CBMPRunner ()

This is the CBMPRunner class constructor (default).

*CBMPRunner::CBMPRunner (CBMPData * bmpData)*

This is the CBMPRunner class constructor that initializes the class variables. It assigns a pointer to CBMPData class.

Parameter:

in,out	*bmpData*	If non-null, a pointer to CBMPData class.

CBMPRunner::~CBMPRunner () [virtual]

This is the CBMPRunner class destructor.

Member Function Documentation

void CBMPRunner::advect (double imat, double svol, double sro, double evol, double ero, double delts, double crrat, double & conc, double & romat)

This function simulates the advection of the pollutants in the water column. It is called by each BMP type that computes the concentration of material present in the BMP and the quantities of material that leave the BMP under an advection process.

Parameters:

	imat	The quantity of material entering during the land simulation interval (lbs/ivl).
	svol	The volume at the beginning of the simulation interval (ft^3).
	sro	The flow rate at the beginning of the simulation interval (cfs).
	evol	The volume at the end of the simulation interval (ft^3).
	ero	The flow rate at the end of the simulation interval (cfs).
	delts	Number of seconds in simulation interval (sec/ivl).
	crrat	The factor used to calculate the weighted volume of outflow based on conditions at start/end of the interval.
in,out	*conc*	The updated concentration in the BMP during the simulation interval (lbs/ft^3).
in,out	*romat*	The total amount of material leaving the BMP during the simulation interval (lbs/ivl).

***void CBMPRunner::bmp_a** (int* **nInfiltMethod**, *int* **nInfil_Index**, *bool* **underdrain_on**, *int* **LANDtimestep**, *int* **BMPtimestep**, *int* **npeople**, *int* **ddays**, *int* **releasetype**, *int* **weirtype**, *int &* **counter**, *double* **oinflow**, *double* **BMParea**, *double* **orifice_dia**, *double* **orificeheight**, *double* **orificecoef**, *double* **weirwidth**, *double* **weirheight**, *double* **weirangle**, *double* **cisternoutflow**, *double* **soildepth**, *double* **soilporosity**, *double* **finalf**, *double* **vegparma**, *double* **holtpar**, *double* **udfinalf**, *double* **udsoildepth**, *double* **udsoilporosity**, *double* **FC**, *double* **WP**, *double* **ETrate**, *double &* **AET**, *double &* **perc**, *double &* **ovolume**, *double &* **ostage**, *double &* **infilt**, *double &* **orifice**, *double &* **weir**, *double &* **osa**, *double &* **ostorage**, *double &* **udout**, *double &* **seepage**, *double &* **pump**, ***BMP_A** ** **pBMP**)*

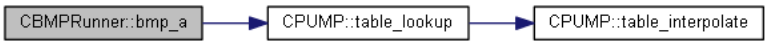
This function simulates the BMP Class A. This BMP class represents the practices that capture upstream drainage at a specific location and can use a combination of detention, infiltration, evaporation, settling, and transformation to manage flow and remove pollutants.

Parameters:

	nInfiltMethod	The infiltration method (0-Green Ampt, 1-Horton, 2-Holtan).
	nInfil_Index	Zero-based index for the unique BMP site.
	underdrain_on	True to enable, false to disable the underdrain.
	LANDtimestep	The land simulation timestep (min).
	BMPtimestep	The BMP simulation timestep (min).
	npeople	The number of people (for Cistern only).
	ddays	The number of dry days (for Rain Barrel only).
	releasetype	The release type (1-Cistern, 2-Rain Barrel, 3-Orifice).
	weirtype	The weir type (1-Rectangular, 2-Triangular).
in,out	*counter*	The counter for counting the dry days (for Rain Barrel only).
	oinflow	The total inflow to the BMP (ft^3 per land timestep).
	BMParea	The BMP surface area (ft^2).
	orifice_dia	The orifice diameter (ft).
	orificeheight	The orifice height (ft).
	orificecoef	The orifice discharge coefficient.
	weirwidth	The weir width (ft).
	weirheight	The weir height (ft).
	weirangle	The weir angle in degrees (for Triangular type).
	cisternoutflow	The cistern outflow (ft^3 per land timestep).
	soildepth	The soil media depth (ft).
	soilporosity	The soil porosity.
	finalf	The saturated hydraulic conductivity (in./hr).
	vegparma	The vegetative coefficient for Holtan infiltration method (0.1–1.0).
	holtpar	The growth index for Holtan infiltration method (0–1).
	udfinalf	The back ground soil infiltration rate (in./hr).
	udsoildepth	The underdrain soil depth (ft).
	udsoilporosity	The underdrain soil porosity.
	FC	The field capacity (in./in.).
	WP	The wilting point (in./in.).
	ETrate	The ET rate (in./land timestep).
in,out	*AET*	The actual ET (cfs).
in,out	*perc*	The percolation (cfs).
in,out	*ovolume*	The current volume in the water column (ft^3).
in,out	*ostage*	The water column depth (ft).
in,out	*infilt*	The infiltrated volume (cfs).
in,out	*orifice*	The orifice outflow (cfs).
in,out	*weir*	The weir outflow (cfs).

in,out	*osa*	The available soil media storage (in.).
in,out	*ostorage*	The available underdrain storage (in.).
in,out	*udout*	The underdrain outflow (cfs).
in,out	*seepage*	The seepage loss (cfs).
in,out	*pump*	The pumped outflow (cfs).
in,out	*pBMP*	If non-null, a pointer to the BMP class A.

This is the call graph for this function:



***void CBMPRunner::bmp_b** (int* **nInfiltMethod**, *int* **nInfil_Index**, *bool* **underdrain_on**, *int* **LANDtimestep**, *int* **BMPtimestep**, *double* **oinflow**, *double* **BMPdepth**, *double* **BMPwidth**, *double* **BMPlength**, *double* **slope1**, *double* **slope2**, *double* **slope3**, *double* **man_n**, *double* **soildepth**, *double* **soilporosity**, *double* **finalf**, *double* **vegparma**, *double* **holtpar**, *double* **udfinalf**, *double* **udsoildepth**, *double* **udsoilporosity**, *double* **FC**, *double* **WP**, *double* **ETrate**, *double &* **AET**, *double &* **perc**, *double &* **ovolume**, *double &* **ostage**, *double &* **infilt**, *double &* **channel**, *double &* **weir**, *double &* **osa**, *double &* **ostorage**, *double &* **udout**, *double &* **seepage**)*

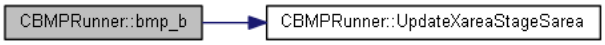
This function simulates the BMP Class B. This BMP class represents the channel type management practices such as swale that does not necessarily detain the flow but provides infiltration, evaporation, and settling to attenuate the flow and remove pollutants.

Parameters:

	nInfiltMethod	The infiltration method (0-Green Ampt, 1-Horton, 2-Holtan).
	nInfil_Index	Zero-based index for the unique BMP site.
	underdrain_on	True to enable, false to disable the underdrain.
	LANDtimestep	The land simulation timestep (min).
	BMPtimestep	The BMP simulation timestep (min).
	oinflow	The total inflow to the BMP (ft^3 per land timestep).
	BMPdepth	The BMP depth to the bed (ft).
	BMPwidth	The BMP bottom width (ft).
	BMPlength	The BMP length (ft).
	slope1	The x-sectional side slope (ft/ft).
	slope2	The x-sectional side slope (ft/ft).
	slope3	The longitudinal slope (ft/ft).
	man_n	Manning's roughness coefficient.
	soildepth	The soil media depth (ft).
	soilporosity	The soil porosity.
	finalf	The saturated hydraulic conductivity (in./hr).
	vegparma	The vegetative coefficient for Holtan infiltration method (0.1–1.0).
	holtpar	The growth index for Holtan infiltration method (0–1).
	udfinalf	The back ground soil infiltration rate (in./hr).
	udsoildepth	The underdrain soil depth (ft).
	udsoilporosity	The underdrain soil porosity.
	FC	The field capacity (in./in.).
	WP	The wilting point (in./in.).
	ETrate	The ET rate (in./land timestep).
in,out	*AET*	The actual ET (cfs).
in,out	*perc*	The percolation (cfs).
in,out	*ovolume*	The current volume in the water column (ft^3).
in,out	*ostage*	The water column depth (ft).

in,out	*infilt*	The infiltrated volume (cfs).
in,out	*channel*	The channel outflow (cfs).
in,out	*weir*	The weir outflow (cfs).
in,out	*osa*	The available soil media storage (in.).
in,out	*ostorage*	The available underdrain storage (in.).
in,out	*udout*	The underdrain outflow (cfs).
in,out	*seepage*	The seepage loss (cfs).

This is the call graph for this function:



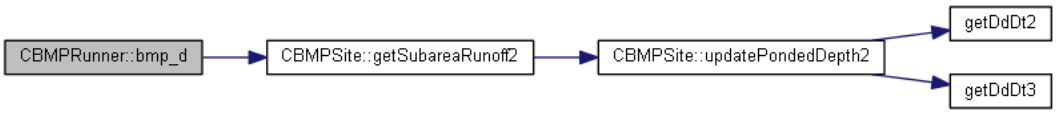
***void CBMPRunner::bmp_d** (**int** nInfiltMethod, **int** nInfil_Index, **int** LANDtimestep, **int** BMPtimestep, **double** oinflow, **double** BMParea, **double** soildepth, **double** soilporosity, **double** finalf, **double** vegparma, **double** holtpar, **double** FC, **double** WP, **double** ETrate, **double &** AET, **double &** ovolume, **double &** ostage, **double &** infilt, **double &** weir, **double &** osa, **double &** seepage, **CBMPSite** * pBMPSite)*

This function simulates the BMP Class D. This BMP class represents the buffer strip that provides infiltration, evaporation, and settling to remove pollutants.

Parameters:

	nInfiltMethod	The infiltration method (0-Green Ampt, 1-Horton, 2-Holtan).
	nInfil_Index	Zero-based index for the unique BMP site.
	LANDtimestep	The land simulation timestep (min).
	BMPtimestep	The BMP simulation timestep (min).
	oinflow	The total inflow to the BMP (ft^3 per land timestep).
	BMParea	The BMP surface area (ft^2).
	soildepth	The soil media depth (ft).
	soilporosity	The soil porosity.
	finalf	The saturated hydraulic conductivity (in./hr).
	vegparma	The vegetative coefficient for Holtan infiltration method (0.1–1.0).
	holtpar	The growth index for Holtan infiltration method (0–1).
	FC	The field capacity (in./in.).
	WP	The wilting point (in./in.).
	ETrate	The ET rate (in./land timestep).
in,out	*AET*	The actual ET (cfs).
in,out	*ovolume*	The current volume in the water column (ft^3).
in,out	*ostage*	The ponded water depth (ft).
in,out	*infilt*	The infiltrated volume (cfs).
in,out	*weir*	The surface outflow (cfs).
in,out	*osa*	The available soil media storage (in.).
in,out	*seepage*	The seepage loss (cfs).
in,out	*pBMPSite*	If non-null, the pointer to the BMP site.

This is the call graph for this function:



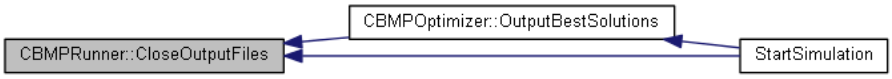
***bool CBMPRunner::CloseOutputFiles** ()*

This function closes an evaluation output file for the assessment points.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:



bool ***CBMPRunner::OpenOutputFiles*** *(const CString &* **runID***, int* **nRunOption***, int* **nRunMode***)*

This function opens an evaluation output file for the assessment points.

Parameters:

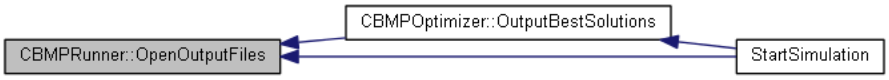
runID	The identifier for the assessment point.
nRunOption	The optimization run option (0-no optimization, 1-minimize cost, 2-cost effectiveness curve).
nRunMode	The simulation run mode (0-baseline, 1-optimize, 2-output, 3-preDev, 4-postDev).

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:

This is the caller graph for this function:



void ***CBMPRunner::RunModel*** *(int* **nRunMode***)*

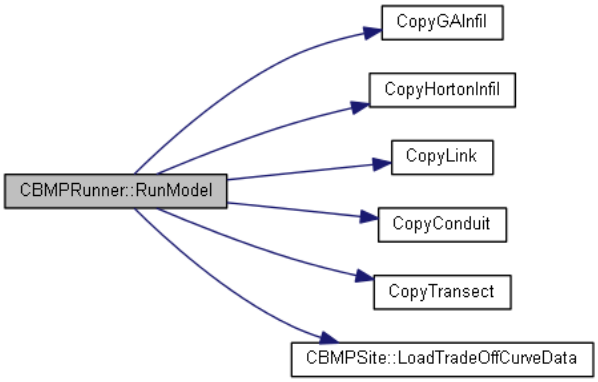
This function executes the BMP simulation module. This module simulates the BMP and conveyance network. It performs the following key steps.

- Gets the inflow time series to the BMP network (Nodes and Links) for the given run mode.
- Calculates the BMP cost on the basis of the BMP dimensions for the given run mode.
- Simulates the BMP network for flow and water quality.
- Calculates the assessment point evaluation factors.
- Outputs the simulation results at the assessment points.

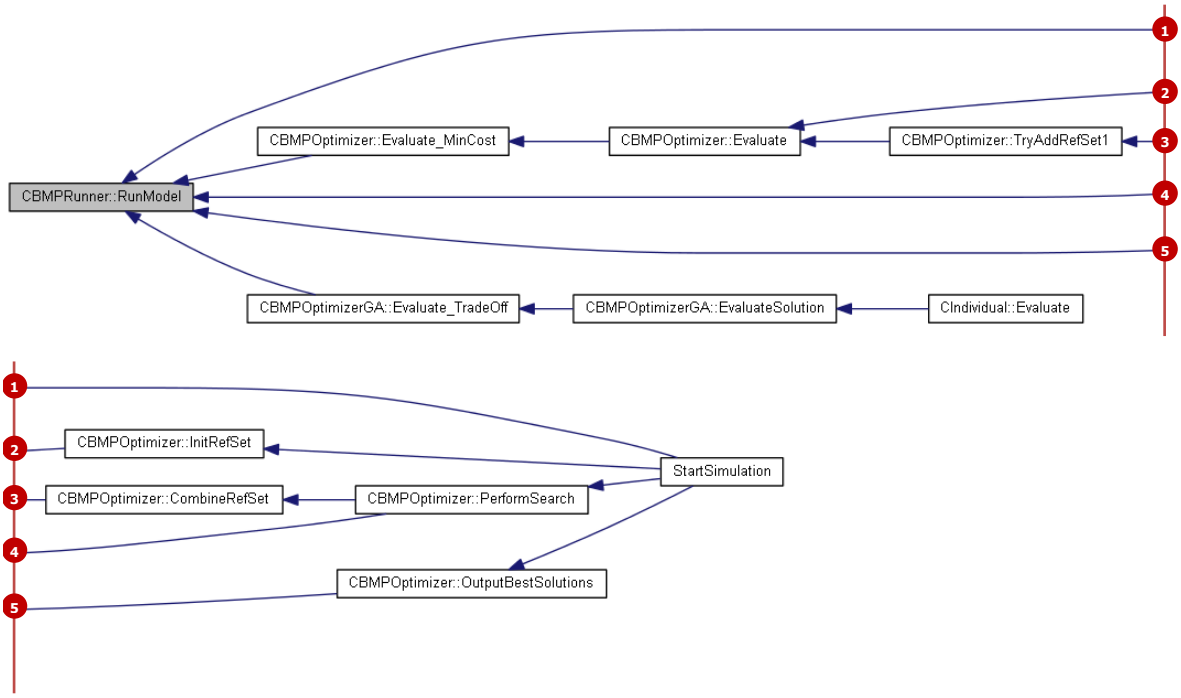
Parameter:

nRunMode	The simulation run mode (0-baseline, 1-optimize, 2-output, 3-preDev, 4-postDev).

This is the call graph for this function:



This is the caller graph for this function:



***void CBMPRunner::UpdateXareaStageSarea** (double* **nvolume**, *double* **vol_max**, *double* **s_area_max**, *double* **BMPdepth**, *double* **BMPwidth**, *double* **BMPlength**, *double* **slope1**, *double* **slope2**, *double &* **x_area**, *double &* **nstage**, *double &* **sur_area**)*

This function updates the water column parameters for BMP Class B. It updates the cross-sectional area, depth, and surface area for the water column in a swale BMP.

Parameters:

	nvolume	The current volume of the water column (ft^3).
	vol_max	The maximum possible volume based on the channel geometry (ft^3).
	s_area_max	The maximum possible surface area based on the channel geometry (ft^2).
	BMPdepth	The BMP depth to the bed (ft).
	BMPwidth	The BMP bottom width (ft).

	BMPlength	The BMP length (ft).
	slope1	The x-sectional side slope (ft/ft).
	slope2	The x-sectional side slope (ft/ft).
in,out	*x_area*	The x-sectional area for the water column (ft^2).
in,out	*nstage*	The water column depth (ft).
in,out	*sur_area*	The water column surface area (ft^2).

This is the caller graph for this function:

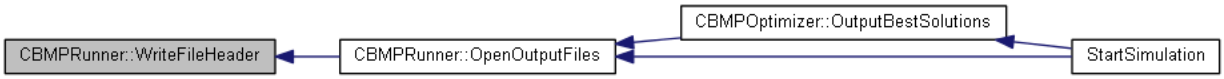
void CBMPRunner::WriteFileHeader *(int* nRunOption*, int* nRunMode*)*

This function writes an evaluation output file header for the assessment points.

Parameters:

nRunOption	The optimization run option (0-no optimization, 1-minimize cost, 2-cost effectiveness curve).
nRunMode	The simulation run mode (0-baseline, 1-optimize, 2-output, 3-preDev, 4-postDev).

This is the caller graph for this function:



Member Data Documentation

*FILE ** CBMPRunner::fp

This is the pointer to the evaluation output file.

double CBMPRunner::lInitRunTime

This is the baseline scenario run time (in milliseconds).

long CBMPRunner::nMaxRun

This is the maximum number of optimization runs.

int CBMPRunner::nRUN_BATCH

This is the flag for batch mode (0-off, 1-on).

int CBMPRunner::optcounter

This is the number of optimization runs.

int CBMPRunner::outcounter

This is the number of ouput runs.

*CBMPData ** CBMPRunner::pBMPData

This is the pointer to CBMPData class.

*CProgressWnd ** CBMPRunner::pWndProgress

This is the pointer to CProgressWnd class. This is used to display and update the progress bar window for the model simulation runs.

COleDateTime ***CBMPRunner::time_i***

This is the time at the beginning of the BMP simulation module.

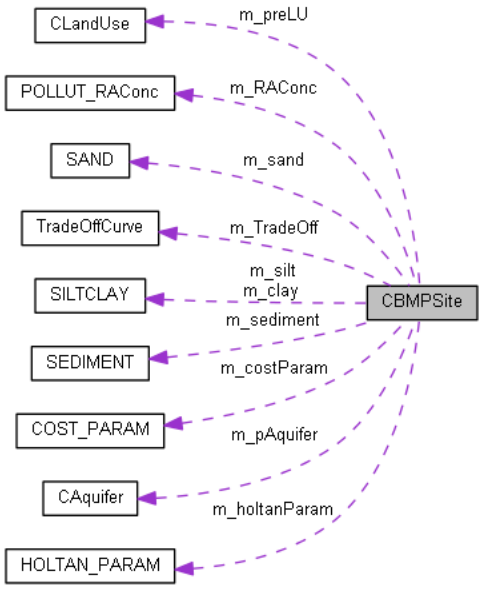
The documentation for this class was generated from the following files:

- BMPRunner.h
- BMPRunner.cpp

CBMPSite Class Reference

#include "BMPSite.h"

Collaboration diagram for CBMPSite:



Public Member Functions

- CBMPSite ()
- CBMPSite (const CString &strID, const CString &strName, const CString &strType, int bmpClass)
- virtual ~CBMPSite ()
- double * GetVariablePointer (CString strVarName)
- double GetBMPArea ()
- bool ReadTradeOffCurveCosts ()
- bool LoadTradeOffCurveData (long deltm, int BPindex, COleDateTime startDate, COleDateTime endDate)
- bool UnLoadTradeOffCurveData (int nBrPtIndex)
- bool LoadWatershedTSData (long deltm, CString BMPSiteID, COleDateTime startDate, COleDateTime endDate, CString strFileName, double *multiplier, int landfg)
- void updatePondedDepth2 (float *depth, float *dt)
- void getSubareaRunoff2 (float inflow, float precip, float evap, float tStep, float *infil, float *depth, float *runoff)

Public Attributes

- CString m_strID
- CString m_strName
- CString m_strType
- bool m_bChecked
- bool m_bUndSwitch
- int m_nInfiltMethod
- int m_nPolRotMethod
- int m_nPolRemMethod
- int m_nInfil_Index
- int m_nBMPClass
- int m_nInterEvent
- double m_lfBMPUnit
- double m_lfDDarea

- double m_lfAccDArea
- double m_lfPerArea
- double m_lfImpArea
- double m_lfSoilDepth
- double m_lfPorosity
- double m_lfFCapacity
- double m_lfWPoint
- double m_lfInfilt
- double m_lfUndDepth
- double m_lfUndVoid
- double m_lfUndInfilt
- double m_lfCost
- double m_lfSurfaceArea
- double m_lfExcavatnVol
- double m_lfSurfStorVol
- double m_lfSoilStorVol
- double m_lfUdrnStorVol
- double m_lfThreshFlow
- double m_lfSiteDArea
- double * m_pDecay
- double * m_pK
- double * m_pCstar
- double * m_pConc
- double * m_pUndRemoval
- void * m_pSiteProp
- CLandUse * m_preLU
- CAquifer * m_pAquifer
- POLLUT_RAConc * m_RAConc
- FILE * m_fileOut
- HOLTAN_PARAM m_holtanParam
- COST_PARAM m_costParam
- CPtrList m_dsbmpsiteList
- CPtrList m_adjustList
- CPtrList m_factorList
- CPtrList m_usbmpsiteList
- CObList m_siteluList
- CObList m_sitepsList
- int m_nBreakPoints
- double m_lfBreakPtID
- CString m_strCostFile
- TradeOffCurve * m_TradeOff
- int m_nQualNum
- long m_nTSNum
- double * m_pDataMixLU
- double * m_pDataPreLU
- COleDateTime m_tmStart
- SEDIMENT m_sediment
- SAND m_sand
- SILTCLAY m_silt
- SILTCLAY m_clay
- queue< double > qFlow

Detailed Description

This class defines the data structure for the given BMP site. The BMP site is a unique location of each BMP class simulated in *SUSTAIN*. This class defines the data structure for hydrology and water quality parameters. It also defines the arrays of pre-developed and post-developed land output timeseries data (described in Data Flow Model section) for the iternal land simulation option. The CBMPData class handles the collection of all BMP sites using CObList from MFC library.

Constructor & Destructor Documentation

CBMPSite::CBMPSite ()

This is the CBMPSite class constructor (default).

CBMPSite::CBMPSite (const CString & strID, const CString & strName, const CString & strType, int bmpClass)

This is the CBMPSite class constructor that initializes the class variables.

Parameters:

strID	The BMP unique identifier.
strName	The BMP name.
strType	The BMP type.
bmpClass	The BMP class type.

CBMPSite::~CBMPSite () [virtual]

This is the CBMPSite class destructor.

Member Function Documentation

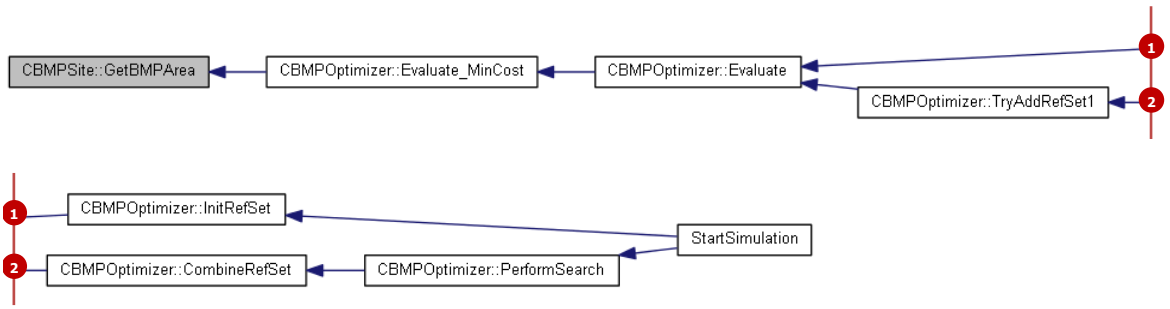
double CBMPSite::GetBMPArea ()

This function computes the BMP surface area.

Returns:

The surface area of the given BMP.

This is the caller graph for this function:



*void CBMPSite::getSubareaRunoff2 (float inflow, float precip, float evap, float tStep, float * infil, float * depth, float * runoff)*

This function computes runoff and losses from a subarea over the current time step.

Parameters:

	inflow	The runoff over subarea (ft/sec).
	precip	The dummy argument (not used)
	evap	The evaporation (ft/sec).
	tStep	The time step (sec/ivl).
in,out	*infil*	The infiltration rate (ft/sec).
in,out	*depth*	The depth of surface runoff (ft).
in,out	*runoff*	The runoff for the current time step (ft/sec).

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:

This is the caller graph for this function:

*double * **CBMPSite::GetVariablePointer** (CString* strVarName*)*

This function gets a pointer to the decision variable.

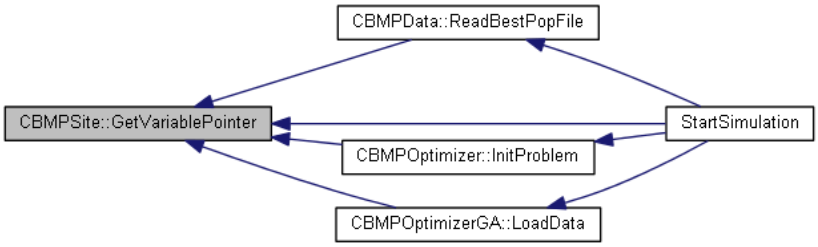
Parameter:

strVarName	The name of the decision variable.

Returns:

Null if it fails, the pointer to the decision variable.

This is the caller graph for this function:



*bool **CBMPSite::LoadTradeOffCurveData** (long* deltm*, int* BPindex*, COleDateTime* startDate*, COleDateTime* endDate*)*

This function reads the time series results for the cost-effectiveness curve solution.

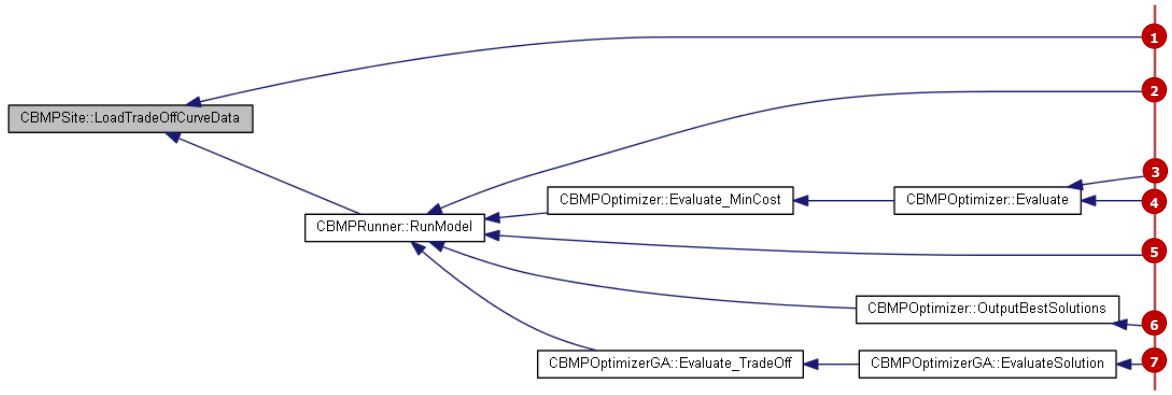
Parameters:

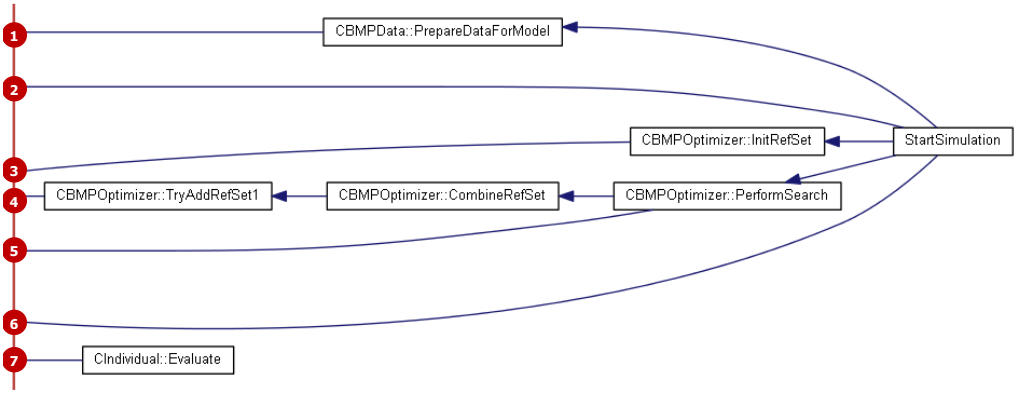
deltm	The simulation timestep in minutes.
BPindex	The index for the break point in the best solution file.
startDate	The start date of the model simulation.
endDate	The end date of the model simulation.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:





***bool CBMPSite::LoadWatershedTSData** (long* **deltm**, *CString* **BMPSiteID**, *COleDateTime* **startDate**, *COleDateTime* **endDate**, *CString* **strFileName**, *double ** **multiplier**, *int* **landfg**)*

This function reads internal land simulation time series data.

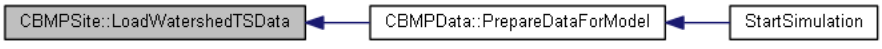
Parameters:

	deltm	The model timestep in minutes.
	BMPSiteID	The BMP site unique identifier.
	startDate	The start date of BMP simulation.
	endDate	The end date of BMP simulation.
	strFileName	The land simulation output file name.
in,out	*multiplier*	If non-null, the multiplier to the time series data.
	landfg	The flag to distinguish the time series file for pre-developed run or baseline run.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:



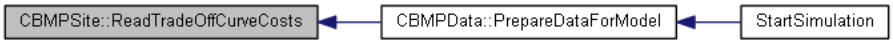
***bool CBMPSite::ReadTradeOffCurveCosts** ()*

This function reads the total cost for the cost-effectiveness curve solution.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:



bool *CBMPSite::UnLoadTradeOffCurveData* *(int* nBrPtIndex*)*

This function releases the assigned memory for the cost-effectiveness curve time series data.

Parameter:

nBrPtIndex	The index for the break point associated with the time series file.

Returns:

True if it succeeds, false if it fails.

void *CBMPSite::updatePondedDepth2* *(float ** depth*, float ** dt*)*

This function computes new ponded depth over subarea after current time step.

Parameters:

in,out	*depth*	The depth of surface runoff (ft).
in,out	*dt*	The time step (sec).

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:

This is the caller graph for this function:

Member Data Documentation

CPtrList *CBMPSite::m_adjustList*

This is the element type of ADJUSTABLE_PARAM in the pointer list.

bool *CBMPSite::m_bChecked*

This is the flag to check for the closed loop when creating the BMP routing list.

bool *CBMPSite::m_bUndSwitch*

This is the switch to turn on the underdrain option (0-off, 1-on).

SILTCLAY *CBMPSite::m_clay*

This is the data structure of SILTCLAY type for sediment transport.

COST_PARAM *CBMPSite::m_costParam*

This is the data structure for BMP cost parameters.

CPtrList ***CBMPSite::m_dsbmpsiteList***

This is the element type in the list DS_BMPSITE.

CPtrList ***CBMPSite::m_factorList***

This is the element type in the list is EVALUATION_FACTOR.

*FILE ** ***CBMPSite::m_fileOut***

This is the file pointer used for outputing the simulation result.

HOLTAN_PARAM CBMPSite::m_holtanParam

This is the data structure for the Holtan infiltration parameters.

double ***CBMPSite::m_lfAccDArea***

This is the accumulative drainage area for the BMP site (acres).

double ***CBMPSite::m_lfBMPUnit***

This is the number of BMP units.

double ***CBMPSite::m_lfBreakPtID***

This is the break point ID for the time series.

double ***CBMPSite::m_lfCost***

This is the BMP cost ($).

double ***CBMPSite::m_lfDDarea***

This is the BMP design drainage area (acres).

double ***CBMPSite::m_lfExcavatnVol***

This is the BMP excavation volume (acre-ft).

double ***CBMPSite::m_lfFCapacity***

This is the soil field capacity (fraction).

double ***CBMPSite::m_lfImpArea***

This is the impervious drainage area for the BMP site (acres).

double ***CBMPSite::m_lfInfilt***

This is the saturated soil infiltration rate (in./hr).

double ***CBMPSite::m_lfPerArea***

This is the pervious drainage area for the BMP site (acres).

double ***CBMPSite::m_lfPorosity***

This is the soil porosity (fraction).

double ***CBMPSite::m_lfSiteDArea***

This is the drainage area for the BMP site (acres).

double ***CBMPSite::m_lfSoilDepth***

This is the BMP substrate soil depth (ft).

double ***CBMPSite::m_lfSoilStorVol***

This is the BMP soil storage volume (acre-ft).

***double* CBMPSite::m_lfSurfaceArea**

This is the BMP surface area (acres).

***double* CBMPSite::m_lfSurfStorVol**

This is the BMP surface storage volume (acre-ft).

***double* CBMPSite::m_lfThreshFlow**

This is the user-defined threshold flow (cfs).

***double* CBMPSite::m_lfUdrnStorVol**

This is the BMP underdrain storage volume (acre-ft).

***double* CBMPSite::m_lfUndDepth**

This is the depth of filter media for underdrain option (ft).

***double* CBMPSite::m_lfUndInfilt**

This is the soil infiltration rate underneath the filter media (in./hr).

***double* CBMPSite::m_lfUndVoid**

This is the filter media voids for the underdrain option (fraction).

***double* CBMPSite::m_lfWPoint**

This is the soil wilting point (fraction).

***int* CBMPSite::m_nBMPClass**

This is the BMP class type (1-class A, 2-class B, 3-class C, 4-class D).

***int* CBMPSite::m_nBreakPoints**

This is the total number of break points on the cost effectiveness curve.

***int* CBMPSite::m_nInfil_Index**

This is an index for the Green-Ampt or Horton infiltration array.

***int* CBMPSite::m_nInfiltMethod**

This is the infiltration method (0-Green-Ampt, 1-Horton, 2-0-Holtan).

***int* CBMPSite::m_nInterEvent**

This is an inter-event duration to define the start and end of an overflow event (hr).

***int* CBMPSite::m_nPolRemMethod**

This is the pollutant removal method (0-1st order decay, 1-Kadlec and Knight method).

***int* CBMPSite::m_nPolRotMethod**

This is the pollutant routing method (1-Completely mixed, >1-number of CSTRs in series).

***int* CBMPSite::m_nQualNum**

This is the total number of constituents in the time series including flow and water quality.

***long* CBMPSite::m_nTSNum**

This is the total number of records for the simulation duration.

***CAquifer ** CBMPSite::m_pAquifer**

This is an aquifer and the actual pointer type is CAquifer.

double** * **CBMPSite::m_pConc

This is an array of concentration for pollutants.

double** * **CBMPSite::m_pCstar

This is an array of background concentration for pollutants (mg/L)

double** * **CBMPSite::m_pDataMixLU

This is the pointer to the time series data for internal land simulation (mixed landuses).

double** * **CBMPSite::m_pDataPreLU

This is the pointer to the time series data for internal land simulation (predeveloped landuses).

double** * **CBMPSite::m_pDecay

This is an array of decay/loss rate for pollutants.

double** * **CBMPSite::m_pK

This is an array of constant rate for pollutants (ft/hr).

CLandUse** * **CBMPSite::m_preLU

This is the pre-developed land use type and the actual pointer type is CLandUse.

void** * **CBMPSite::m_pSiteProp

This is the BMP type pointer for the BMP site.

double** * **CBMPSite::m_pUndRemoval

This is an array of removal rate for underdrain pollutants.

POLLUT_RAConc** * **CBMPSite::m_RAConc

This is the data structure of running average concentration evaluation factor.

SAND CBMPSite::m_sand

This is the data structure of sand parameters for sediment transport.

SEDIMENT CBMPSite::m_sediment

This is the data structure of general parameters for sediment transport.

SILTCLAY CBMPSite::m_silt

This is the data structure of silt parameters for sediment transport.

CObList** **CBMPSite::m_siteluList

This is the list for all site land use associated with this BMP site.

CObList** **CBMPSite::m_sitepsList

This is the list for all site point sources associated with this BMP site.

CString** **CBMPSite::m_strCostFile

This is the cost-effectiveness curve solution file.

CString** **CBMPSite::m_strID

This is the BMP unique identifier.

CString** **CBMPSite::m_strName

This is the descriptive BMP name.

CString ***CBMPSite::m_strType***

This is the BMP type.

COleDateTime ***CBMPSite::m_tmStart***

This is the start date of the time series data.

TradeOffCurve * ***CBMPSite::m_TradeOff***

This is an array of break point time series files.

CPtrList ***CBMPSite::m_usbmpsiteList***

This is the pointer list for all upstream BMP sites that flow into this BMP site directly, element type in the list is DS_BMPSITE.

queue< double > ***CBMPSite::qFlow***

This is an array that stores the previous day values for flow (cfs).

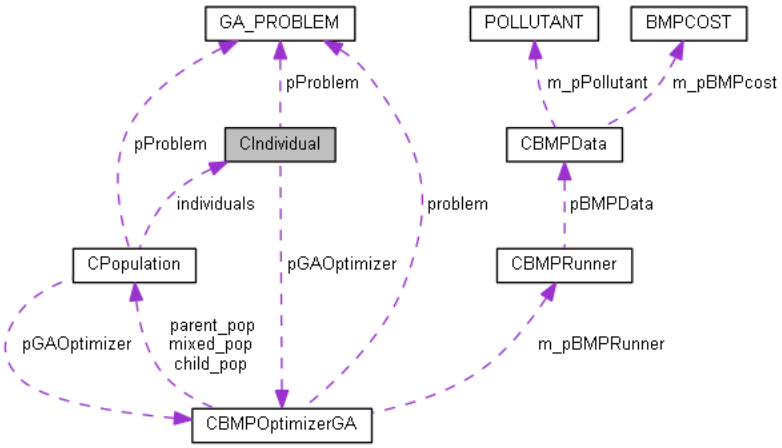
The documentation for this class was generated from the following files:

- BMPSite.h
- BMPSite.cpp

CIndividual Class Reference

#include "Individual.h"

Collaboration diagram for CIndividual:



Public Member Functions

- CIndividual ()
- CIndividual (GA_PROBLEM *pProb, CBMPOptimizerGA *pOptimizer)
- virtual ~CIndividual ()
- void Allocate (GA_PROBLEM *pProb, CBMPOptimizerGA *pOptimizer)
- void Initialize ()
- void Evaluate ()
- void Mutate ()
- void CopyFrom (const CIndividual &ind)
- int CheckDominance (const CIndividual &ind)

Public Attributes

- bool validSolution
- int rank
- double crowd_dist
- double constr_violation
- double * xreal
- double * obj
- double * BMPcost
- double * constr
- GA_PROBLEM * pProblem
- CBMPOptimizerGA * pGAOptimizer

Detailed Description

This class defines functions for NSGA-II operations that are carried out to individual solutions. The functions are construction and destruction of the CIndividual class, allocating memory space for the GA optimization problem (CIndividual::Allocate), initializing the optimization (CIndividual::Initialize), evaluating individual solutions (CIndividual::Evaluate), mutating individual solutions, copying individual solutions, and checking the dominance of one individual solution over another (CIndividual::CheckDominance).

Constructor & Destructor Documentation

CIndividual::CIndividual ()

This is the CIndividual class constructor (default).

*CIndividual::CIndividual (GA_PROBLEM * pProb, CBMPOptimizerGA * pOptimizer)*

This is the CIndividual class constructor that initializes the class variables. It assigns a pointer to CBMPOptimizerGA class.

Parameters:

in,out	*pProb*	If non-null, a pointer to GA_PROBLEM class.
in,out	*pOptimizer*	If non-null, a pointer to CBMPOptimizerGA class.

This is the call graph for this function:

CIndividual::~CIndividual () [virtual]

This is the CIndividual class destructor.

Member Function Documentation

*void CIndividual::Allocate (GA_PROBLEM * pProb, CBMPOptimizerGA * pOptimizer)*

This function allocates memory for arrays that are associated with each individual solution. The arrays include real number decision variables, the optimization targets, the BMP costs, and the optimization constraints.

Parameters:

in,out	*pProb*	If non-null, a pointer to GA_PROBLEM class.
in,out	*pOptimizer*	If non-null, a pointer to CBMPOptimizerGA class.

This is the caller graph for this function:

int CIndividual::CheckDominance (const CIndividual & ind)

This function performs the dominance checking between the current individual and the subject individual (*ind*). The dominance checking is performed on two aspects: violation of constraints and value of optimization target.

Parameter:

in,out	*ind*	A subject individual that is to be compared with the current individual for dominance.

Returns:

1 if the current individual dominates the subject individual
-1 if the subject individual dominates the current individual
0 if the two individuals do not dominate each other

This is the caller graph for this function:

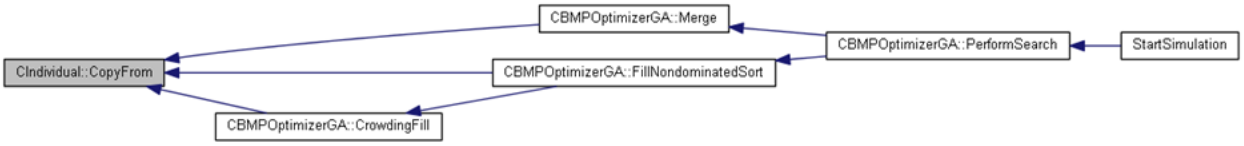
void CIndividual::CopyFrom (const CIndividual & ind)

This function creates a copy of the subject individual (ind). The real number decision variables, objective target values, BMP costs, and the constraint values are all copied from the subject individual to the current individual.

Parameter:

in,out	*ind*	A subject individual that is to be copied to the current individual.

This is the caller graph for this function:



void CIndividual::Evaluate ()

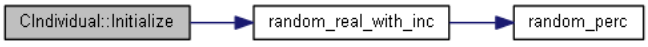
This function evaluates individual solutions through a call of the CBMPOptimizerGA::EvaluateSolution function. The constraint violations for an individual solution are also calculated.

This is the call graph for this function:

void CIndividual::Initialize ()

This function initializes a real number individual for the optimization process. The BMP costs are also initialized as zero.

This is the call graph for this function:



void CIndividual::Mutate ()

This function carries out polynomial mutation to individual solutions.

This is the call graph for this function:

Member Data Documentation

*double * CIndividual::BMPcost*

This function carries out polynomial mutation to individual solutions.

double * CIndividual::constr

This is an array of constraint values for an individual solution.

double CIndividual::constr_violation

This is the number of constraint violations by an individual solution.

double CIndividual::crowd_dist

This is the crowding distance of an individual solution.

double * CIndividual::obj

This is an array of objective target values calculated for an individual solution.

CBMPOptimizerGA * CIndividual::pGAOptimizer

This is a pointer to CBMPOptimizerGA class.

GA_PROBLEM * CIndividual::pProblem

This is a pointer to GA_PROBLEM class.

int CIndividual::rank

This is the rank of an individual solution in a population.

bool CIndividual::validSolution

This is an indicator of whether the evaluation of an individual solution is successful.

double * CIndividual::xreal

This is an array of real numbers that represents the decision variables during the optimization process.

The documentation for this class was generated from the following files:

- Individual.h
- Individual.cpp

CLandUse Class Reference

#include "LandUse.h"

Public Member Functions

- CLandUse ()
- virtual ~CLandUse ()
- bool LoadLanduseTSData (long deltm, COleDateTime startDate, COleDateTime endDate, double *multiplier)

Public Attributes

- int m_nID
- int m_nType
- int m_nQualNum
- long m_nTSNum
- double m_lfsand_fr
- double m_lfsilt_fr
- double m_lfclay_fr
- double * m_pData
- CString m_strLanduse
- CString m_strFileName
- COleDateTime m_tmStart

Detailed Description

This is the data structure for CLandUse class.

Constructor & Destructor Documentation

CLandUse::CLandUse ()

This is the CLandUse class constructor (default).

CLandUse::~CLandUse () [virtual]

This is the CLandUse class destructor.

Member Function Documentation

bool **CLandUse::LoadLanduseTSData** *(long* **deltm***, COleDateTime* **startDate***, COleDateTime* **endDate***, double ** **multiplier***)*

This function reads the land use time series data.

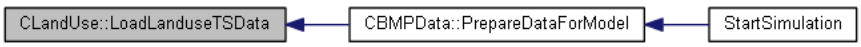
Parameters:

	deltm	The timestep in minutes.
	startDate	The start date of model simulation.
	endDate	The end date of model simulation.
in,out	*multiplier*	If non-null, the multiplier to the time series pollutants loading.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:



Member Data Documentation

***double* CLandUse::m_lfclay_fr**

This is the fraction of clay in the total sediment loading from this land use type.

***double* CLandUse::m_lfsand_fr**

This is the fraction of sand in the total sediment loading from this land use type.

***double* CLandUse::m_lfsilt_fr**

This is the fraction of silt in the total sediment loading from this land use type.

***int* CLandUse::m_nID**

This is the land use type unique identifier.

***int* CLandUse::m_nQualNum**

This is the number of pollutants.

***long* CLandUse::m_nTSNum**

This is the number of records in the land output time series file.

***int* CLandUse::m_nType**

This is the land use type (0-pevious, 1-impevious).

***double ** CLandUse::m_pData**

This is an array of time series data for this land use type.

***CString* CLandUse::m_strFileName**

This is the full path of the time series file for this land use type.

***CString* CLandUse::m_strLanduse**

This is the land use type name.

***COleDateTime* CLandUse::m_tmStart**

This is the start date (first data record) in the time series file.

The documentation for this class was generated from the following files:

- LandUse.h
- LandUse.cpp

COST_PARAM Struct Reference

#include "BMPSite.h"

Public Attributes

- double m_lfLinearCost
- double m_lfAreaCost
- double m_lfTotalVolumeCost
- double m_lfMediaVolumeCost
- double m_lfUnderDrainVolumeCost
- double m_lfConstantCost
- double m_lfPercentCost
- double m_lfLengthExp
- double m_lfAreaExp
- double m_lfTotalVolExp
- double m_lfMediaVolExp
- double m_lfUDVolExp

Detailed Description

This is the data structure of BMP cost parameters.

Member Data Documentation

double COST_PARAM::m_lfAreaCost

This is the cost per unit area of the BMP structure ($/ft^2).

double COST_PARAM::m_lfAreaExp

This is the cost exponent value of the BMP area variable for economy of the scale (default = 1).

double COST_PARAM::m_lfConstantCost

This is the constant cost ($).

double COST_PARAM::m_lfLengthExp

This is the cost exponent value of the BMP length variable for economy of the scale (default = 1).

double COST_PARAM::m_lfLinearCost

This is the cost per unit length of the BMP structure ($/ft).

double COST_PARAM::m_lfMediaVolExp

This is the cost exponent value of the BMP soil media volume variable for economy of the scale (default = 1).

double COST_PARAM::m_lfMediaVolumeCost

This is the cost per unit volume of the soil media ($/ft^3).

double COST_PARAM::m_lfPercentCost

This is the cost in percentage of all other cost (%).

double COST_PARAM::m_lfTotalVolExp

This is the cost exponent value of the BMP total volume variable for economy of the scale (default = 1).

double COST_PARAM::m_lfTotalVolumeCost

This is the cost per unit total volume of the BMP structure ($/ft^3).

double ***COST_PARAM::m_lfUDVolExp***

This is the cost exponent value of the BMP underdrain media volume variable for economy of the scale (default = 1).

double ***COST_PARAM::m_lfUnderDrainVolumeCost***

This is the cost per unit volume of the under drain structure ($/ft^3).

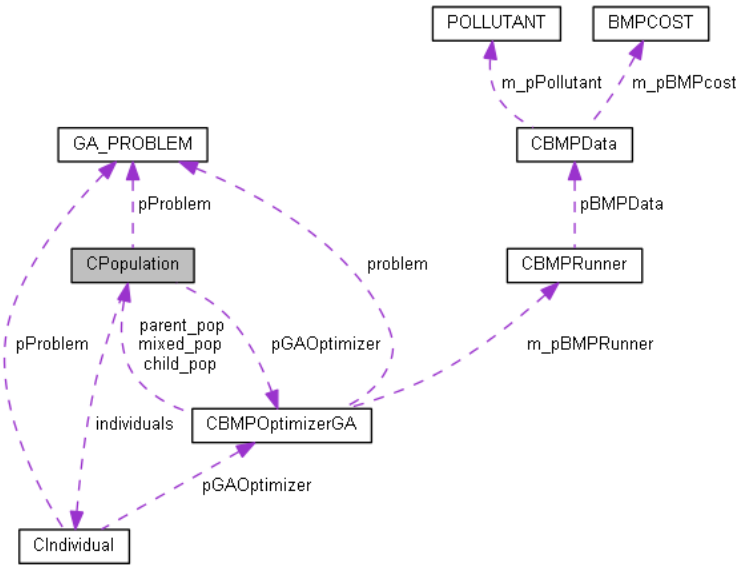
The documentation for this struct was generated from the following file:

- BMPSite.h

CPopulation Class Reference

#include "Population.h"

Collaboration diagram for CPopulation:



Public Member Functions

- CPopulation ()
- CPopulation (int nsize, GA_PROBLEM *pProblem, CBMPOptimizerGA *pOptimizer)
- virtual ~CPopulation ()
- int GetBestSolutionIndex ()
- int GetNextBestSolutionIndex (double prevMinCost)
- void Initialize ()
- void Evaluate ()
- void Mutate ()
- void AssignRankAndCrowdingDistance ()
- void AssignCrowdingDistanceList (void *list, int front_size)
- void AssignCrowdingDistance (int *dist, int **obj_array, int front_size)
- void AssignCrowdingDistanceIndices (int c1, int c2)
- void QuickSortFront (int objcount, int *obj_array, int obj_array_size)
- void QuickSortFrontImpl (int objcount, int *obj_array, int left, int right)
- void QuickSortDist (int *dist, int front_size)
- void QuickSortDistImpl (int *dist, int left, int right)
- void ReportIndividualToFile (FILE *fp, int index)
- void ReportAllToFile (FILE *fp)
- void ReportBestToFile (FILE *fp)

Public Attributes

- int nSize
- GA_PROBLEM * pProblem
- CBMPOptimizerGA * pGAOptimizer
- CIndividual * individuals

Detailed Description

This class defines functions for NSGA-II operations carried out at the population level during the optimization process. The functions consist of include construction and destruction of the CPopulation

class, identifying the best and the second-best individual solutions in a population (CPopulation::GetBestSolutionIndex, CPopulation::GetNextBestSolutionIndex), initializing individuals in a population (CPopulation::Initialize), evaluating individual solutions in a population (CPopulation::Evaluate), mutating individual solutions in a population (CPopulation::Mutate), and to assign rank and crowding distances for individuals in a population (CPopulation::AssignRankAndCrowdingDistance).

When assigning rank and crowding distance for individuals, the population is first divided into different fronts through the non-dominated sorting (CIndividual::CheckDominance), along with the rank of each individual. Subsequently, the crowding distances between individuals in each front are also calculated (CPopulation::AssignCrowdingDistanceList, CPopulation::AssignCrowdingDistance). In preparing the crowding distance calculation, the individuals in each front are first sorted by the optimization objectives (CPopulation::QuickSortFront, CPopulation::QuickSortFrontImpl). The class also includes functions for sorting a population on the basis of the crowding distance (CPopulation::QuickSortDist, CPopulation::QuickSortDistImpl).

The population class also includes functions to write the solutions into output files. The output files can include single individual solution (CPopulation::ReportIndividualToFile), all individual solutions in a population (CPopulation::ReportAllToFile), or only the best solutions identified (CPopulation::ReportBestToFile).

Constructor & Destructor Documentation

CPopulation::CPopulation ()

This is the CPopulation class constructor (default).

CPopulation::CPopulation (int nsize, *GA_PROBLEM* * pProblem, *CBMPOptimizerGA* * pOptimizer*)*

This is the CPopulation class constructor that initializes the class variables. It assigns a pointer to GA_PROBLEM class and CBMPOptimizerGA class.

Parameters:

	nsize	The population size.
in,out	*pProblem*	If non-null, a pointer to GA_PROBLEM class.
in,out	*pOptimizer*	If non-null, a pointer to CBMPOptimizerGA class.

CPopulation::~CPopulation () [virtual]

This is the CPopulation class destructor.

Member Function Documentation

void CPopulation::AssignCrowdingDistance (int * dist, *int* ** obj_array, *int* front_size*)*

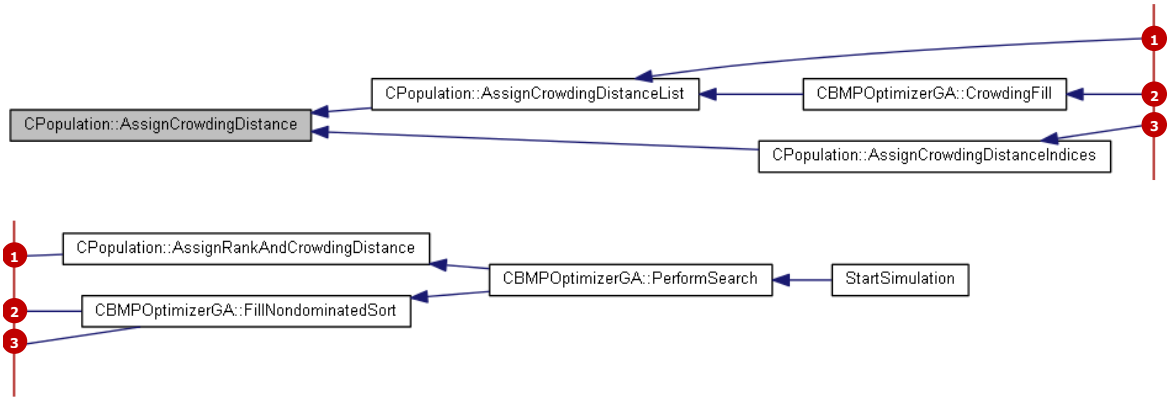
This function assigns crowding distances to individuals in a front for specific objectives. Before the crowding distances are calculated, the front is first sorted according to the objectives through CPopulation::QuickSortFront.

Parameters:

	front_size	The size of the front.
in,out	*dist*	An array of crowding distances of individual solutions in the front
in,out	*obj_array*	An array of the objective values of individual solutions in the front

This is the call graph for this function:

This is the caller graph for this function:



void CPopulation::AssignCrowdingDistanceIndices (int c1, int c2)

This function assigns crowding distances to individuals that are between the denoted starting and ending indices.

Parameters:

c1	The index of the starting individual for the crowding distance calculation.
c2	The index of the ending individual for the crowding distance calculation.

This is the call graph for this function:

This is the caller graph for this function:

*void CPopulation::AssignCrowdingDistanceList (void * list, int front_size)*

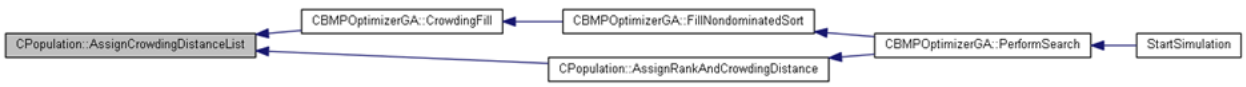
This function assigns crowding distances to individuals that are in an array (list) that resides in a front with certain size (front_size). This function calls the previously defined CPopulation::AssignCrowdingDistance.

Parameters:

	front_size	The size of the front in which that the array resides.
in,out	*list*	The array that contains the individuals to have their crowding distances calculated.

This is the call graph for this function:

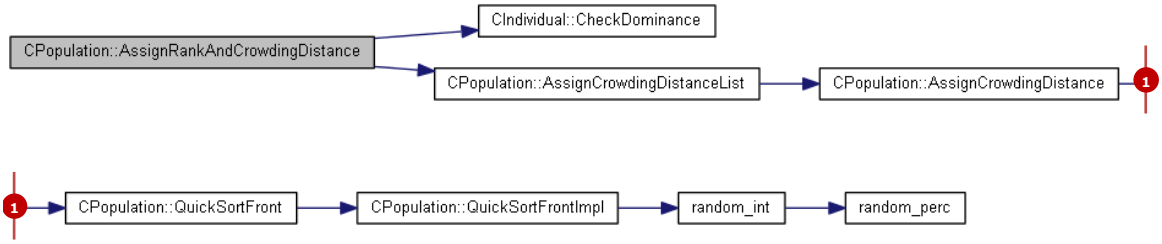
This is the caller graph for this function:



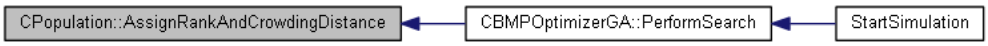
void CPopulation::AssignRankAndCrowdingDistance ()

This function assigns the rank and crowding distances to all individuals in the population. The ranking is calculated through the non-dominated sorting process, and the crowding distance is calculated through a call of CPopulation::AssignCrowdingDistanceList.

This is the call graph for this function:



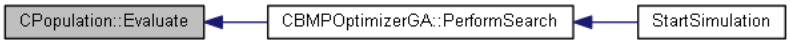
This is the caller graph for this function:



void CPopulation::Evaluate ()

This function evaluates all individual solutions in the population. The evaluation of individual solutions is carried out through a call of CIndividual::Evaluate, which then calls CBMPOptimizerGA::EvaluateSolution.

This is the caller graph for this function:



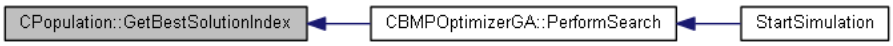
int CPopulation::GetBestSolutionIndex ()

This function retrieves the index of the best individual solution in the current population. The solution with the lowest total cost is regarded as the best solution in the current population.

Returns:

The index of the best solution in the current population.

This is the caller graph for this function:



int CPopulation::GetNextBestSolutionIndex (double prevMinCost*)*

This function retrieves the index of the individual solution with a total cost that is the lowest and at the same time only higher than a pre-specified cost value (prevMinCost).

Parameter:

prevMinCost	The pre-specified cost value that is used to search within the current population for the individual that has a lowest cost while being larger than this specified cost value.

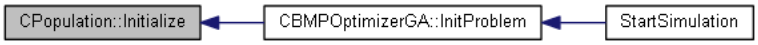
Returns:

The index of the individual solution that has the lowest cost while being larger than the pre-specified cost value.

void CPopulation::Initialize ()

This function initializes individual solutions in the population. This function calls the function of CIndividual::Initialize.

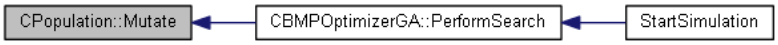
This is the caller graph for this function:



void CPopulation::Mutate ()

This function performs mutation to individual solutions in the population. This function calls the function of CIndividual::Mutate.

This is the caller graph for this function:



*void CPopulation::QuickSortDist (int * dist, int front_size)*

This function quickly sorts of a population using the crowding distances of individual solutions. This function calls for the function of CPopulation::QuickSortDistImpl.

Parameters:

	front_size	The size of the front to be sorted.
in,out	*dist*	The array that stores the crowding distances of individual solutions.

This is the call graph for this function:

This is the caller graph for this function:

*void CPopulation::QuickSortDistImpl (int * dist, int left, int right)*

This function quickly sorts a population according to the crowding distance, with the population being defined by the left and right indices of the individual solutions. A recursion of the function is implemented to help expedite the sorting process.

Parameters:

	left	The starting index of the individual in the population to be sorted.
	right	The ending index of the individual in the population to be sorted.
in,out	*dist*	An array that stores the crowding distances of the individual solutions.

This is the call graph for this function:

This is the caller graph for this function:

void ***CPopulation::QuickSortFront*** *(int* objcount*, int ** obj_array*, int* obj_array_size*)*

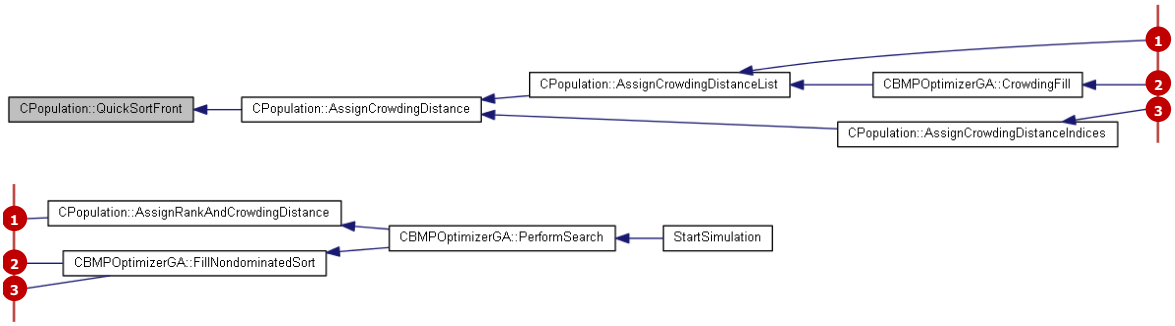
This function quickly sorts a population using an objective. This function calls the function of CPopulation::QuickSortFrontImpl.

Parameters:

	objcount	The index of objectives.
	obj_array_size	The size of the population to be sorted.
in,out	*obj_array*	The array that stores the objectives.

This is the call graph for this function:

This is the caller graph for this function:



void ***CPopulation::QuickSortFrontImpl*** *(int* objcount*, int ** obj_array*, int* left*, int* right*)*

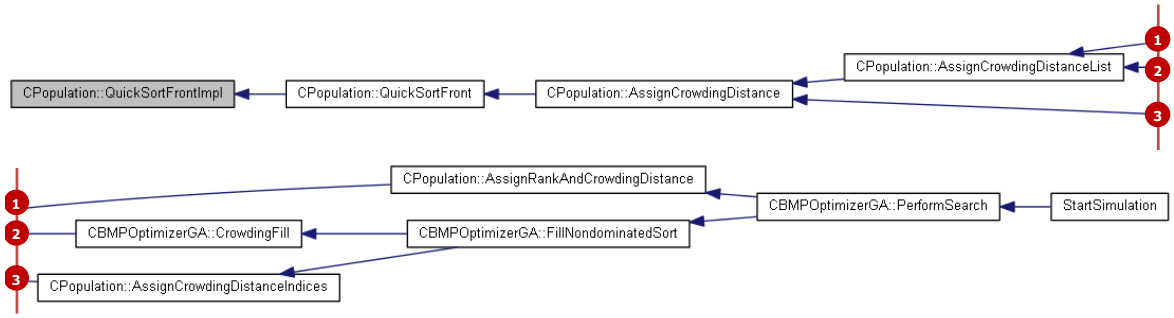
This function quickly sorts a population on an objective, with the population being defined with the left and the right indices of the individual solutions. A recursion of the function is implemented to help expedite the sorting process.

Parameters:

	objcount	The index of objectives.
	left	The starting index of the individual in the population to be sorted.
	right	The ending index of the individual in the population to be sorted.
in,out	*obj_array*	The array that stores the objectives.

This is the call graph for this function:

This is the caller graph for this function:



void ***CPopulation::ReportAllToFile*** *(FILE * fp)*

This function reports all individual solutions in a population to an output file. This function calls the function of CPopulation::ReportIndividualToFile.

Parameter:

in,out	*fp*	A pointer to the designated output file for storing all solutions.

This is the call graph for this function:

void ***CPopulation::ReportBestToFile*** *(FILE * fp)*

This function reports only the best solutions to an output file. The best solution is defined as the one without constraint violation and has a rank equal to one.

Parameter:

in,out	*fp*	A pointer to the designated output file for storing the best solution.

This is the call graph for this function:

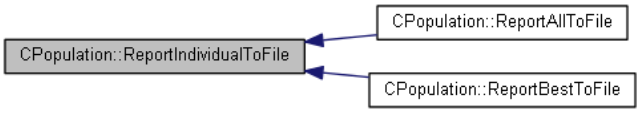
void ***CPopulation::ReportIndividualToFile*** *(FILE * fp, int index)*

This function reports an individual solution to an output file.

Parameters:

	index	The index of the individual solution in the population.
in,out	*fp*	A pointer to the designated output file for storing the individual solution.

This is the caller graph for this function:



Member Data Documentation

CIndividual * CPopulation::individuals

This is a pointer to the CIndividual class.

int ***CPopulation::nSize***

This is the population size.

CBMPOptimizerGA * CPopulation::pGAOptimizer

This is a pointer to the CBMPOptimizerGA class.

GA_PROBLEM * CPopulation::pProblem

This is a pointer to the GA_PROBLEM class.

The documentation for this class was generated from the following files:

- Population.h
- Population.cpp

CPUMP Class Reference

#include "Pump.h"

Public Member Functions

- CPUMP ()
- CPUMP (CString strPCurveID, int nRecords, double *lfDepth, double *lfFlow)
- virtual ~CPUMP ()
- double table_interpolate (double x, double x1, double y1, double x2, double y2)
- double table_lookup (double x)

Public Attributes

- CString m_strPCurveID
- int m_nRecords
- double * m_lfDepth
- double * m_lfFlow

Detailed Description

This class defines the data structure for the pump option.

Constructor & Destructor Documentation

CPUMP::CPUMP ()

This is the CPUMP class constructor (default).

CPUMP::CPUMP (CString strPCurveID, *int* nRecords, *double ** lfDepth, *double ** lfFlow*)*

This is the CPUMP class constructor that initializes the class variables on the basis of the user inputs.

Parameters:

	strPCurveID	The identifier for the unique curve ID as a string.
	nRecords	The total number of records for the pump curve.
in,out	*lfDepth*	If non-null, the array of water depth (ft).
in,out	*lfFlow*	If non-null, the array of pump flow rate (cfs).

CPUMP::~CPUMP () [virtual]

This is the CPUMP class destructor.

Member Function Documentation

double CPUMP::table_interpolate (double x, *double* x1, *double* y1, *double* x2, *double* y2*)*

This function interpolates a y value for a given x value.

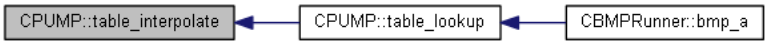
Parameters:

x	The value being interpolated.
x1	The values on either side of x.
y1	The values on either side of y.
x2	The values on either side of x.
y2	The values on either side of y.

Returns:

The y value corresponding to the x value.

This is the caller graph for this function:



double ***CPUMP::table_lookup*** *(double x)*

This function retrieves the y-value corresponding to an x-value in a table using interploation if necessary.

Parameter:

x	The value being looked up.

Returns:

The y-value.

This is the call graph for this function:

This is the caller graph for this function:

Member Data Documentation

*double ** ***CPUMP::m_lfDepth***

This is an array of water depth (ft).

*double ** ***CPUMP::m_lfFlow***

This is an array of the pump flow rate (cfs).

int ***CPUMP::m_nRecords***

This is the total number of records for the given pump curve.

CString ***CPUMP::m_strPCurveID***

This is the unique identifier for the pump curve as a string.

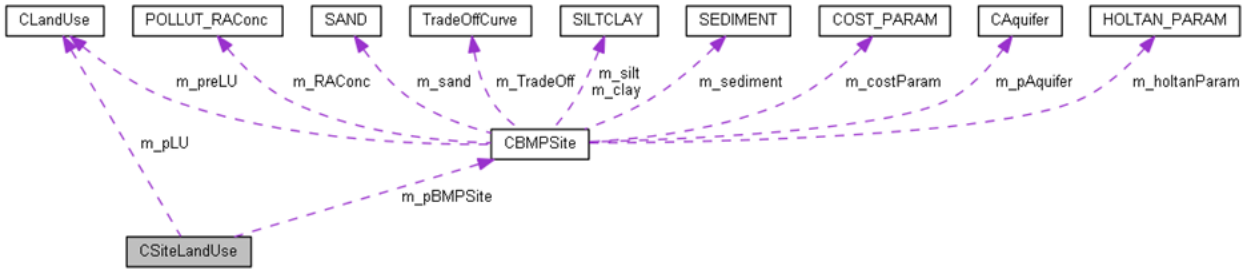
The documentation for this class was generated from the following files:

- Pump.h
- Pump.cpp

CSiteLandUse Class Reference

#include "SiteLandUse.h"

Collaboration diagram for CSiteLandUse:



Public Member Functions

- CSiteLandUse ()
- CSiteLandUse (CLandUse *pLU, CBMPSite *pBMPSite, double lfArea)
- virtual ~CSiteLandUse ()

Public Attributes

- double m_lfArea
- CLandUse * m_pLU
- CBMPSite * m_pBMPSite

Detailed Description

The data structure for the CSiteLandUse class.

Constructor & Destructor Documentation

CSiteLandUse::CSiteLandUse ()

This is the CSiteLandUse class constructor (default).

*CSiteLandUse::CSiteLandUse (CLandUse * pLU, CBMPSite * pBMPSite, double lfArea)*

This is the CSiteLandUse class constructor that initializes the class variables. It assigns a pointer to CLandUse and CBMPSite classes.

Parameters:

	lfArea	The drainage area for this land use type for the BMP site.
in,out	*pLU*	If non-null, a pointer to CLandUse class.
in,out	*pBMPSite*	If non-null, a pointer to CBMPSite class.

CSiteLandUse::~CSiteLandUse () [virtual]

This is the CSiteLandUse class destructor.

Member Data Documentation

double CSiteLandUse::m_lfArea

This is the drainage area for this land use type for the BMP site.

*CBMPSite * CSiteLandUse::m_pBMPSite*

This is a pointer to the BMP site object of type CBMPSite class.

CLandUse* * *CSiteLandUse::m_pLU

This is a pointer to this land use type of type CLandUse class.

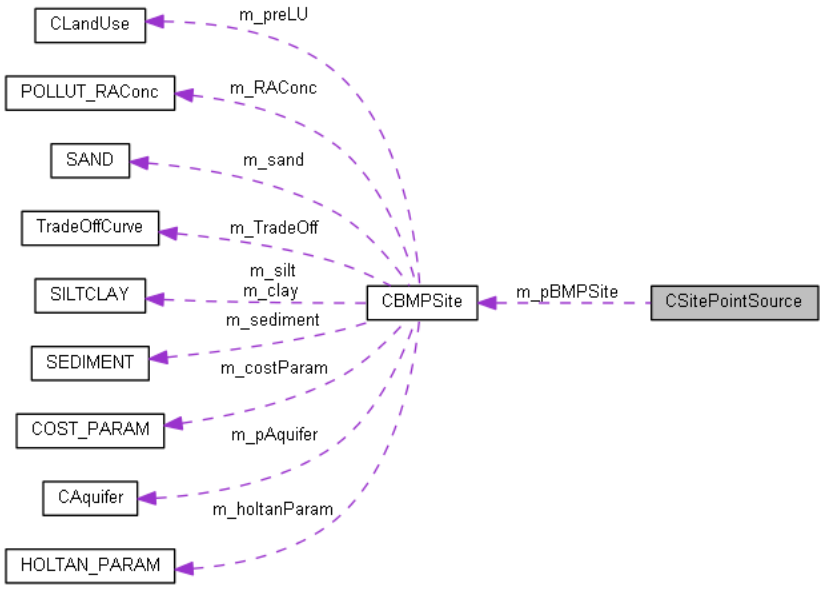
The documentation for this class was generated from the following files:

- SiteLandUse.h
- SiteLandUse.cpp

CSitePointSource Class Reference

#include "SitePointSource.h"

Collaboration diagram for CSitePointSource:



Public Member Functions

- CSitePointSource ()
- virtual ~CSitePointSource ()
- bool LoadPointsourceTSData (long deltm, COleDateTime startDate, COleDateTime endDate, double *multiplier)

Public Attributes

- CString m_strPSDesc
- CString m_strPSFile
- COleDateTime m_tmStart
- int m_nID
- int m_nQualNum
- long m_nTSNum
- double m_lfMult
- double m_lfSand
- double m_lfSilt
- double m_lfClay
- double * m_pDataPS
- CBMPSite * m_pBMPSite

Detailed Description

This is the data structure class for the point source.

Constructor & Destructor Documentation

CSitePointSource::CSitePointSource ()

This is the CSitePointSource class constructor (default).

CSitePointSource::~CSitePointSource () [virtual]

This is the CSitePointSource class destructor.

Member Function Documentation

bool CSitePointSource::LoadPointsourceTSData *(long* deltm, *COleDateTime* startDate, *COleDateTime* endDate, *double ** multiplier*)*

This function reads the point source time series data.

Parameters:

	deltm	The timestep in minutes.
	startDate	The start date of model simulation.
	endDate	The end date of model simulation.
in,out	*multiplier*	If non-null, the multiplier to the time series.

Returns:

True if it succeeds, false if it fails.

This is the caller graph for this function:

Member Data Documentation

double CSitePointSource::m_lfClay

This is the fraction of total sediment that is clay.

double CSitePointSource::m_lfMult

This is the multiplier to the time series file.

double CSitePointSource::m_lfSand

This is the fraction of total sediment that is sand.

double CSitePointSource::m_lfSilt

This is the fraction of total sediment that is silt.

int CSitePointSource::m_nID

This is the unique point source identifier.

int CSitePointSource::m_nQualNum

This is the number of pollutants.

long CSitePointSource::m_nTSNum

This is the number of records in the time series file.

*CBMPSite ** CSitePointSource::m_pBMPSite

This is the pointer to the associated BMPSite.

*double ** CSitePointSource::m_pDataPS

This is an array storing the point source time series data.

CString CSitePointSource::m_strPSDesc

This is the point source description.

CString ***CSitePointSource::m_strPSFile***

This is the point source file name.

COleDateTime ***CSitePointSource::m_tmStart***

This is the start date of model simulation.

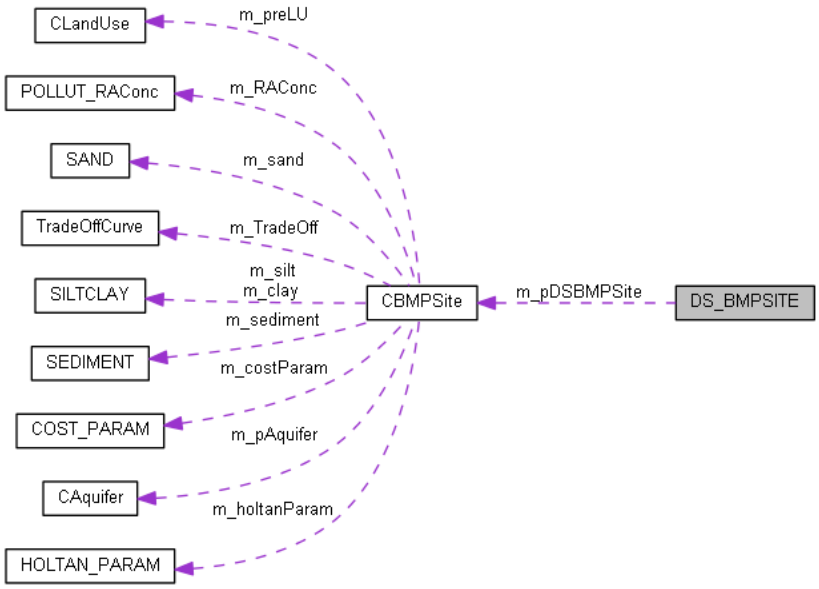
The documentation for this class was generated from the following files:

- SitePointSource.h
- SitePointSource.cpp

DS_BMPSITE Struct Reference

#include "BMPSite.h"

Collaboration diagram for DS_BMPSITE:



Public Attributes

- int m_nOutletType
- CBMPSite * m_pDSBMPSite

Detailed Description

This is the data structure class for downstream BMP site.

Member Data Documentation

int DS_BMPSITE::m_nOutletType

This is the outflow type (1-total outflow, 2-weir outflow, 3-orifice or channel outflow, and 4-underdrain outflow).

*CBMPSite * DS_BMPSITE::m_pDSBMPSite*

This is the pointer to a downstream BMP site.

The documentation for this struct was generated from the following file:

- BMPSite.h

EVALUATION_FACTOR Struct Reference

#include "BMPSite.h"

Public Attributes

- CString m_strFactor
- int m_nFactorGroup
- int m_nFactorType
- int m_nCalcMode
- int m_nCalcDays
- double m_lfThreshold
- double m_lfTarget
- double m_lfLowerTarget
- double m_lfUpperTarget
- double m_lfInit
- double m_lfCurrent
- double m_lfPreDev
- double m_lfPostDev

Detailed Description

This is the data structure class for BMP evaluation factor parameters.

Member Data Documentation

double EVALUATION_FACTOR::m_lfCurrent

This is the current value of the evaluation factor for the optimization run.

double EVALUATION_FACTOR::m_lfInit

This is the initial value of the evaluation factor for the baseline run.

double EVALUATION_FACTOR::m_lfLowerTarget

This is the lower target value for the cost-effectiveness curve.

double EVALUATION_FACTOR::m_lfPostDev

This is the value of the evaluation factor for the post-developed run (developed land uses and no BMPs).

double EVALUATION_FACTOR::m_lfPreDev

This is the value of the evaluation factor for the pre-developed run (single land use and no BMPs).

double EVALUATION_FACTOR::m_lfTarget

This is the target value of evaluation factor.

double EVALUATION_FACTOR::m_lfThreshold

This is the flow threshold if the evaluation factor is *flow exceeding frequency*.

double EVALUATION_FACTOR::m_lfUpperTarget

This is the upper target value for the cost-effectiveness curve.

int EVALUATION_FACTOR::m_nCalcDays

If the evaluation factor type is 3, it is the maxmimum number of days. If the evaluation factor type is -3, it is the flow threshold (cfs).

int EVALUATION_FACTOR::m_nCalcMode

This is the evaluation factor calculation mode; 1 = %, 2 = Scale, 3 = Value.

int ***EVALUATION_FACTOR::m_nFactorGroup***

This is the evaluation factor group; a negative number for flow, a positive number for a pollutant.

int ***EVALUATION_FACTOR::m_nFactorType***

This is the evaluation factor type: -1 = Annual Average Flow Volume (ft^3/yr), -2 = Peak Discharge Flow (cfs), -3 = Flow Exceeding frequency (cfs), 1 = Annual Average Load (lb/yr), 2 = Annual Average Concentration (mg/L), 3 = Maximum #days Average Concentraion (mg/L).

CString ***EVALUATION_FACTOR::m_strFactor***

This is the evaluation factor name, e.g., FlowVolume or TSSLoad.

The documentation for this struct was generated from the following file:

- BMPSite.h

GA_PROBLEM Struct Reference

#include "Global.h"

Public Attributes

- int popsize
- int ngen
- int nobj
- int ncon
- int nBMPtype
- int nreal
- int nrealmut
- int nrealcross
- double eta_c
- double eta_m
- double pcross_real
- double pmut_real
- double * min_realvar
- double * max_realvar
- double * inc_realvar

Detailed Description

This is the NSGA-II optimization class structure. The structure defines basic parameters for carrying out the optimization process. The parameters are the population size; number of generations to be evaluated; the number of objectives; the number of constraints; the number of BMP types; the number of real number variables; the number of real number mutations; the number of real number crossovers; the distribution index for crossover; the distribution index for mutation; the probability for real number crossover; the probability for real number mutation; the arrays of minimum, maximum, and increment values for each real number variable.

Member Data Documentation

double ***GA_PROBLEM::eta_c***

This is the distribution index for crossover. The distribution index must be a positive value.

double ***GA_PROBLEM::eta_m***

This is the distribution index for mutation. The distribution index must be a positive value.

*double ** ***GA_PROBLEM::inc_realvar***

This is the array of increment values associated with real number variables in each individual solution.

*double ** ***GA_PROBLEM::max_realvar***

This is the array of maximum real number variable values in each individual solution.

*double ** ***GA_PROBLEM::min_realvar***

This is the array of minimum real number variable values in each individual solution.

int ***GA_PROBLEM::nBMPtype***

This is the BMP type in an individual solution.

int ***GA_PROBLEM::ncon***

This is the number of constraints for the optimization process.

int ***GA_PROBLEM::ngen***

This is the number of generations to be carried out for the optimization process.

int GA_PROBLEM::nobj

This is the number of objectives for the optimization process.

int GA_PROBLEM::nreal

This is the number of real number variables for the optimization problem.

int GA_PROBLEM::nrealcross

This is the number of real number variable crossovers carried out during the optimization process.

int GA_PROBLEM::nrealmut

This is the number of real number mutations carried out during the optimization process.

double GA_PROBLEM::pcross_real

This is the probability for carrying out crossover for real number variables. The probability is a value between 0 and 1 (including bounds).

double GA_PROBLEM::pmut_real

This is the probability for carrying out mutation for real number variables. The probability is a value between 0 and 1 (including bounds).

int GA_PROBLEM::popsize

This is the population size during the optimization process. The population size must be a multiple of 4.

The documentation for this struct was generated from the following file:

- Global.h

HOLTAN_PARAM Struct Reference

#include "BMPSite.h"

Public Attributes

- double m_lfVegA
- double m_lfGrowth [12]

Detailed Description

This is the data structure class for the Holtan infiltration method.

Member Data Documentation

*double **HOLTAN_PARAM::m_lfGrowth**[12]*

This is an array of monthly growth index used in the Holtan infiltration equation.

*double **HOLTAN_PARAM::m_lfVegA***

This is the vegetative coefficient A used in the Holtan infiltration equation.

The documentation for this struct was generated from the following file:

- BMPSite.h

POLLUT_RAConc Class Reference

#include "BMPSite.h"

Public Member Functions

- POLLUT_RAConc ()
- virtual ~POLLUT_RAConc ()

Public Attributes

- int m_nRDays
- double * m_lfRFlow
- double * m_lfRLoad

Detailed Description

This is the data structure class for running average pollutant concentration.

Constructor & Destructor Documentation

POLLUT_RAConc::POLLUT_RAConc () [inline]

This is the POLLUT_RAConc class constructor (default).

POLLUT_RAConc::~POLLUT_RAConc () [inline, virtual]

This is the POLLUT_RAConc class destructor.

Member Data Documentation

*double * POLLUT_RAConc::m_lfRFlow*

This is an array for storing the running average flow rate.

*double * POLLUT_RAConc::m_lfRLoad*

This is an array for storing the running average pollutant load.

int POLLUT_RAConc::m_nRDays

This is the maximum number of running average days.

The documentation for this class was generated from the following file:

- BMPSite.h

POLLUTANT Struct Reference

#include "BMPData.h"

Public Attributes

- CString m_sName
- int m_nID
- int m_nSedfg
- int m_nSedQual
- double m_lfMult
- double m_lfsand_qfr
- double m_lfsilt_qfr
- double m_lfclay_qfr

Detailed Description

This is data structure class for pollutants.

Member Data Documentation

double POLLUTANT::m_lfclay_qfr

This is the fraction of adsorbed pollutant on clay.

double POLLUTANT::m_lfMult

This is the multiplier to covert the pollutant load into pounds.

double POLLUTANT::m_lfsand_qfr

This is the fraction of adsorbed pollutant on sand.

double POLLUTANT::m_lfsilt_qfr

This is the fraction of adsorbed pollutant on silt.

int POLLUTANT::m_nID

This is the unique pollutant identifier.

int POLLUTANT::m_nSedfg

This is the flag to distinguish the pollutant as sediment.

int POLLUTANT::m_nSedQual

This is the flag to distinguish the adsorbed pollutant on sediment.

CString POLLUTANT::m_sName

This is the name description of the pollutant type.

The documentation for this struct was generated from the following file:

- BMPData.h

PUMP_CONTROL Struct Reference

#include "BMPSite.h"

Public Attributes

- int m_nPumpFlag
- bool m_bIsPumping
- double m_lfDepth_ON
- double m_lfDepth_OFF
- CString m_strPCurveID

Detailed Description

This is data structure class for the pump parameters.

Member Data Documentation

bool PUMP_CONTROL::m_bIsPumping

This is the flag for pumping status (0-false, 1-true).

double PUMP_CONTROL::m_lfDepth_OFF

This is the depth (ft) at which pump is stopped.

double PUMP_CONTROL::m_lfDepth_ON

This is the depth (ft) at which pump is started.

int PUMP_CONTROL::m_nPumpFlag

This is the pump option (0-no, 1-yes).

CString PUMP_CONTROL::m_strPCurveID

This is the unique pump curve identifier (continuous string).

The documentation for this struct was generated from the following file:

- BMPSite.h

SAND Struct Reference

#include "BMPSite.h"

Public Attributes

- double m_lfD
- double m_lfW
- double m_lfRHO
- double m_lfKSAND
- double m_lfEXPSND

Detailed Description

This is the data structure class for non-cohesive sediment parameters.

Member Data Documentation

double SAND::m_lfD

This is the effective diameter of the transported sand particles (in.).

double SAND::m_lfEXPSND

This is the exponent in the sandload power function formula.

double SAND::m_lfKSAND

This is the coefficient in the sandload power function formula.

double SAND::m_lfRHO

This is the density of the sand particles (lb/ft^3).

double SAND::m_lfW

This is the corresponding fall velocity in still water (in./sec).

The documentation for this struct was generated from the following file:

- BMPSite.h

SCATTER_SEARCH Struct Reference

#include "Global.h"

Public Attributes

- int b1
- int b2
- int PSize
- int iter
- int n_var
- int last_combine
- int new_elements
- int * iter1
- int * iter2
- int * order1
- int * order2
- int * evaOrders
- int ** ranges
- double digits
- double * high
- double * low
- double * inc
- double * value1
- double * value2
- double * evaValues
- double ** evaSolutions
- double ** refSet1
- double ** refSet2

Detailed Description

This is the data structure class for the Scatter Search optimization option.

Member Data Documentation

int ***SCATTER_SEARCH::b1***

This is the number of solutions in reference set 1.

int ***SCATTER_SEARCH::b2***

This is the number of solutions in reference set 2.

double ***SCATTER_SEARCH::digits***

This is the number of digits after the decimal point.

*int ** ***SCATTER_SEARCH::evaOrders***

This is an array storing the sorting order of solutions.

*double *** ***SCATTER_SEARCH::evaSolutions***

This is an array storing solutions of all generations.

*double ** ***SCATTER_SEARCH::evaValues***

This is an array storing the values of solutions.

*double ** ***SCATTER_SEARCH::high***

This is an array of high bounds of decision variables.

*double * SCATTER_SEARCH::inc*

This is an array of increments of decision variables.

int SCATTER_SEARCH::iter

This is the number of iterations of the combine reference set process.

*int * SCATTER_SEARCH::iter1*

This is an array of index for the solutions in reference set 1 indicating whether it was used in the last combining reference set process.

*int * SCATTER_SEARCH::iter2*

This is an array of index for the solutions in reference set 2 indicating whether it was used in the last combining reference set process.

int SCATTER_SEARCH::last_combine

This is the last number of iterations of the combine reference set process.

*double * SCATTER_SEARCH::low*

This is an array of low bound of the decision variables.

int SCATTER_SEARCH::n_var

This is the number of decision variables.

int SCATTER_SEARCH::new_elements

This is the new solution flag. A value of 1 indicates a solution was added to the reference set, 0 indicates no new solution was added.

*int * SCATTER_SEARCH::order1*

This is an array of index for the order of solutions in reference set 1.

*int * SCATTER_SEARCH::order2*

This is an array of index for the order of solutions in reference set 2.

int SCATTER_SEARCH::PSize

This is the population size, i.e., number of solutions in one population.

*int ** SCATTER_SEARCH::ranges*

This is an array storing the number of solutions in a specific range of a decision variable.

*double ** SCATTER_SEARCH::refSet1*

This is an array of solutions for reference set 1.

*double ** SCATTER_SEARCH::refSet2*

This is an array of solutions for reference set 2.

*double * SCATTER_SEARCH::value1*

This is an array of values of solutions in reference set 1.

*double * SCATTER_SEARCH::value2*

This is an array of values of solutions in reference set 2.

The documentation for this struct was generated from the following file:

- Global.h

SEDIMENT Struct Reference

#include "BMPSite.h"

Public Attributes

- double m_lfBEDWID
- double m_lfBEDDEP
- double m_lfBEDPOR
- double m_lfSAND_FRAC
- double m_lfSILT_FRAC
- double m_lfCLAY_FRAC

Detailed Description

This is the data structure class for sediment parameters.

Member Data Documentation

double SEDIMENT::m_lfBEDDEP

This is the initial sediment bed depth that is available for scour.

double SEDIMENT::m_lfBEDPOR

This is the sediment bed porosity.

double SEDIMENT::m_lfBEDWID

This is the sediment bed width that is available for scour.

double SEDIMENT::m_lfCLAY_FRAC

This is the fraction of sediment in bed that is clay.

double SEDIMENT::m_lfSAND_FRAC

This is the fraction of sediment in bed that is sand.

double SEDIMENT::m_lfSILT_FRAC

This is the fraction of sediment in bed that is silt.

The documentation for this struct was generated from the following file:

- BMPSite.h

SILTCLAY Struct Reference

#include "BMPSite.h"

Public Attributes

- double m_lfD
- double m_lfW
- double m_lfRHO
- double m_lfTAUCD
- double m_lfTAUCS
- double m_lfM

Detailed Description

This is the data structure class for cohesive sediment parameters.

Member Data Documentation

*double **SILTCLAY::m_lfD***

This is the effective diameter of the transported silt/clay particles (in.).

*double **SILTCLAY::m_lfM***

This is the erodibility coefficient of the sediment (lb/ft^2/day).

*double **SILTCLAY::m_lfRHO***

This is the density of the silt/clay particles (lb/ft^3).

*double **SILTCLAY::m_lfTAUCD***

This is the critical bed shear stress for deposition (lb/ft^2).

*double **SILTCLAY::m_lfTAUCS***

This is the critical bed shear stress for scour (lb/ft^2).

*double **SILTCLAY::m_lfW***

This is the corresponding fall velocity in still water (in./sec).

The documentation for this struct was generated from the following file:

- BMPSite.h

TradeOffCurve Class Reference

#include "BMPSite.h"

Public Member Functions

- TradeOffCurve ()
- virtual ~TradeOffCurve ()

Public Attributes

- int m_nID
- int m_nQualNum
- long m_nTSNum
- double m_lfMult
- double m_lfCost
- double m_lfSand
- double m_lfSilt
- double m_lfClay
- double * m_pDataBrPt
- CString m_strBrPtFile
- COleDateTime m_tmStart

Detailed Description

This is the data structure class for the cost-effectiveness curve parameters.

Constructor & Destructor Documentation

TradeOffCurve::TradeOffCurve () [inline]

This is the TradeOffCurve class constructor (default).

TradeOffCurve::~TradeOffCurve () [inline, virtual]

This is the TradeOffCurve class destructor.

Member Data Documentation

double ***TradeOffCurve::m_lfClay***

This is the fraction of total sediment that is clay.

double ***TradeOffCurve::m_lfCost***

This is the total cost associated with the tier-1 solution.

double ***TradeOffCurve::m_lfMult***

This is the multiplier to the time series file.

double ***TradeOffCurve::m_lfSand***

This is the fraction of total sediment that is sand.

double ***TradeOffCurve::m_lfSilt***

This is the fraction of total sediment that is silt.

int ***TradeOffCurve::m_nID***

This is the break point identifier.

int ***TradeOffCurve::m_nQualNum***

This is the number of pollutants in the time series file.

long ***TradeOffCurve::m_nTSNum***

This is the number of records in the time series file.

*double ** ***TradeOffCurve::m_pDataBrPt***

This is the pointer to the break point data array.

CString ***TradeOffCurve::m_strBrPtFile***

This is the time series file name for the break point on the cost-effectiveness curve.

COleDateTime ***TradeOffCurve::m_tmStart***

This is the DATE data type for the start time.

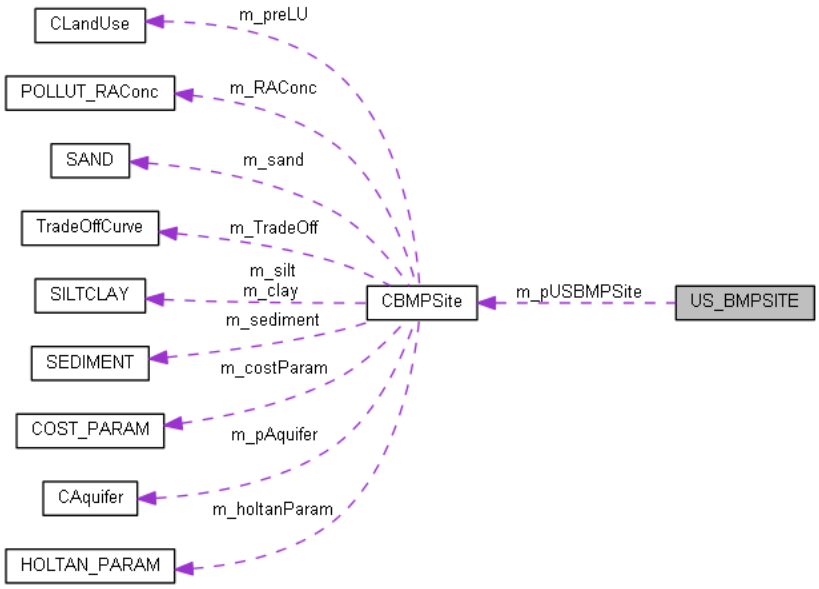
The documentation for this class was generated from the following file:

- BMPSite.h

US_BMPSITE Struct Reference

#include "BMPSite.h"

Collaboration diagram for US_BMPSITE:



Public Attributes

- int m_nOutletType
- CBMPSite * m_pUSBMPSite

Detailed Description

This is the data structure class for the upstream BMP site.

Member Data Documentation

int US_BMPSITE::m_nOutletType

This is the outflow type (1-total outflow, 2-weir outflow, 3-orifice or channel outflow, and 4-underdrain outflow).

*CBMPSite * US_BMPSITE::m_pUSBMPSite*

This is the pointer to a upstream BMP site.

The documentation for this struct was generated from the following file:

- BMPSite.h

File Documentation

Aquifer.cpp File Reference

#include "stdafx.h"
#include "Aquifer.h"

Detailed Description

This file implements the CAquifer class.

Aquifer.h File Reference

Classes

- class CAquifer

Detailed Description

This is header file for the CAquifer class.

BMPData.cpp File Reference

```
#include "stdafx.h"
#include <direct.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "LandUse.h"
#include "BMPSite.h"
#include "SiteLandUse.h"
#include "SitePointSource.h"
#include "BMPData.h"
#include "BMPRunner.h"
#include "StringToken.h"
#include "Global.h"
#include "Aquifer.h"
#include "Pump.h"
```

Functions

- void InitializeGAInfil (int nNum)
- void CopyGAInfil (TGrnAmpt *Source, TGrnAmpt *Target)
- void InitializeHortonInfil (int nNum)
- void CopyHortonInfil (THorton *Source, THorton *Target)
- void InitializeLinkConduitTransect (int nNum)
- void CopyLink (TLink *Source, TLink *Target, int nPollutant)
- void CopyConduit (TConduit *Source, TConduit *Target)
- void CopyTransect (TTransect *Source, TTransect *Target)
- int FindObIndexFromList (CObList &list, CObject *ob)
- void Validate_Conduit (int nNum)

Detailed Description

This file implements the CBMPData class.

Global Function Documentation

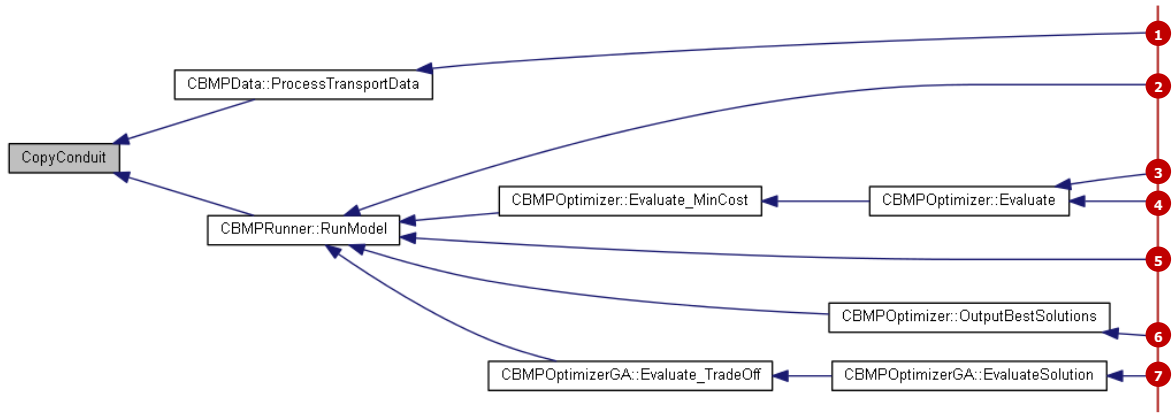
*CopyConduit (TConduit * Source, TConduit * Target)*

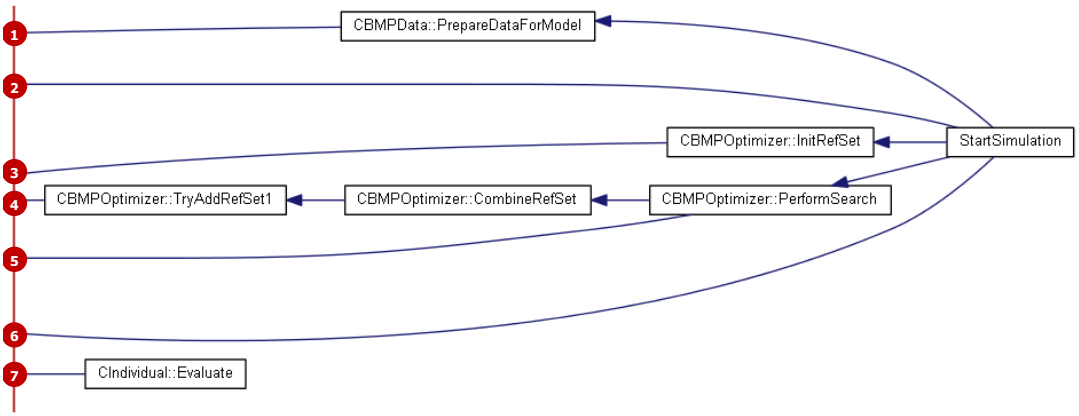
This function assigns the user-defined conduit parameters to another conduit object.

Parameters:

Source	The source data structure.
Target	The target data structure.

This is the caller graph for this function:





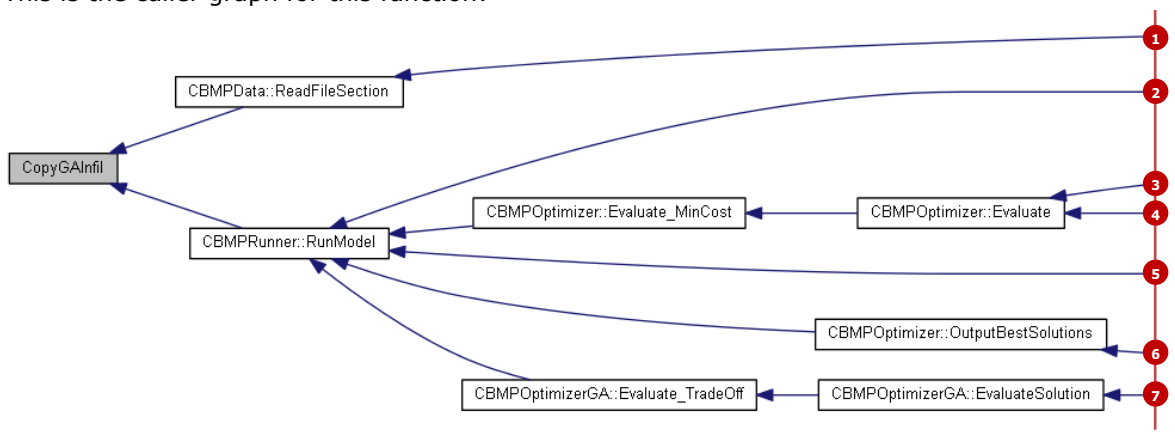
*CopyGAInfil (TGrnAmpt * Source, TGrnAmpt * Target)*

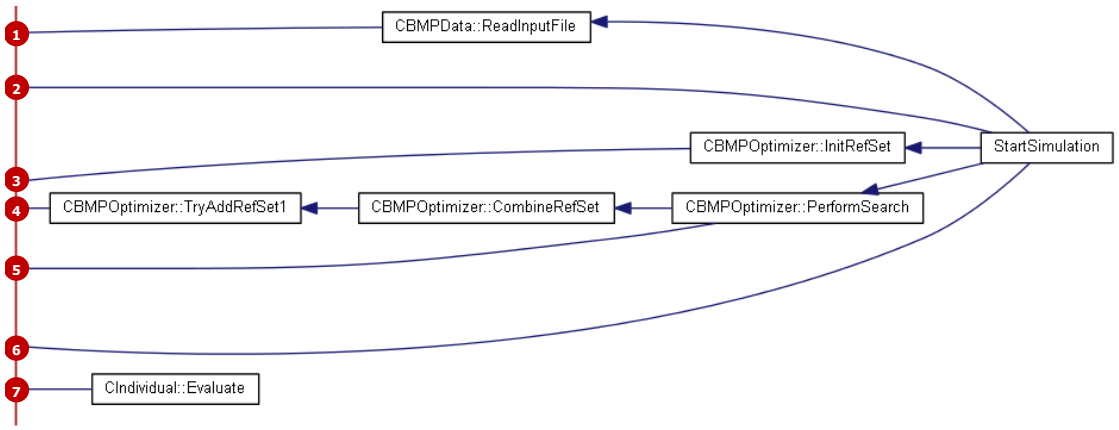
This function creates a copy of the user-defined Green-Ampt infiltration parameters.

Parameters:

Source	The source data structure.
Target	The target data structure.

This is the caller graph for this function:





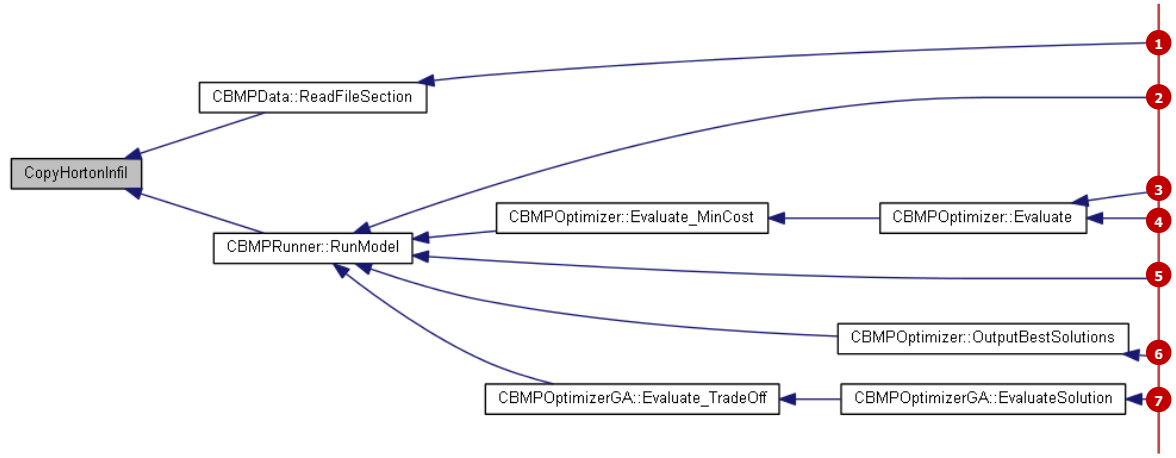
*CopyHortonInfil (THorton * Source, THorton * Target)*

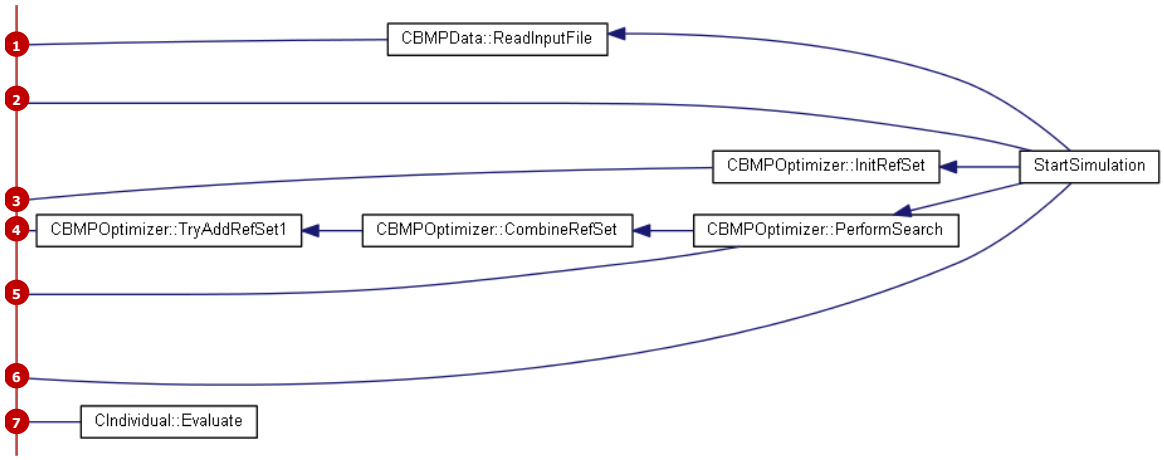
This function creates a copy of the user-defined Horton infiltration parameters.

Parameters:

Source	The source data structure.
Target	The target data structure.

This is the caller graph for this function:





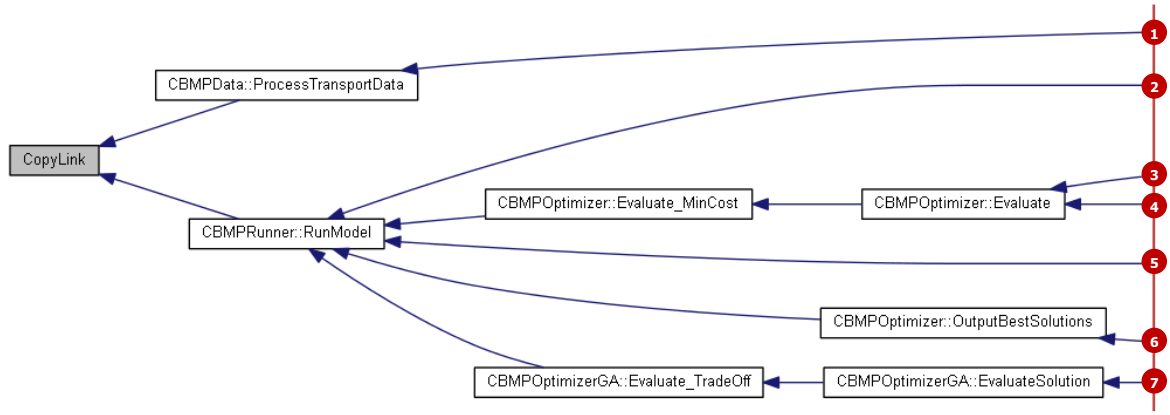
CopyLink *(TLink * Source, TLink * Target, int nPollutant)*

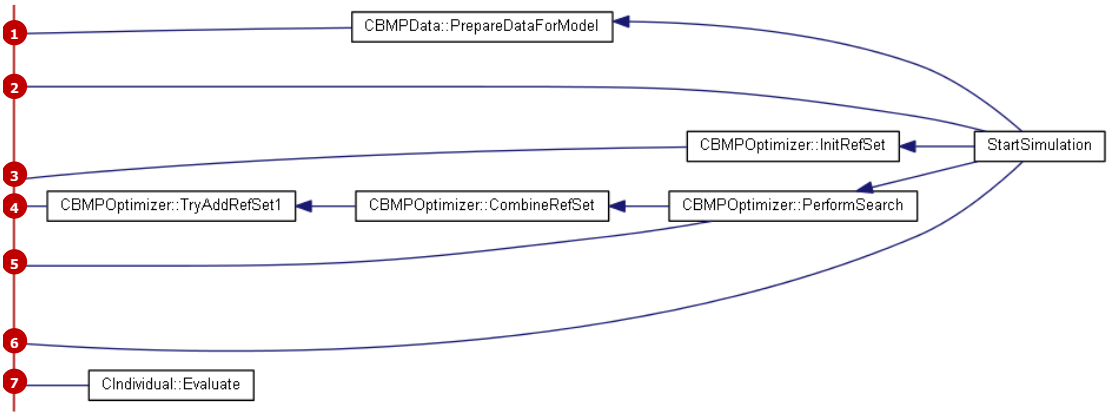
This function creates a copy of the user-defined link parameters.

Parameters:

Source	The source data structure.
Target	The target data structure.
nPollutant	The number of user-defined pollutants.

This is the caller graph for this function:





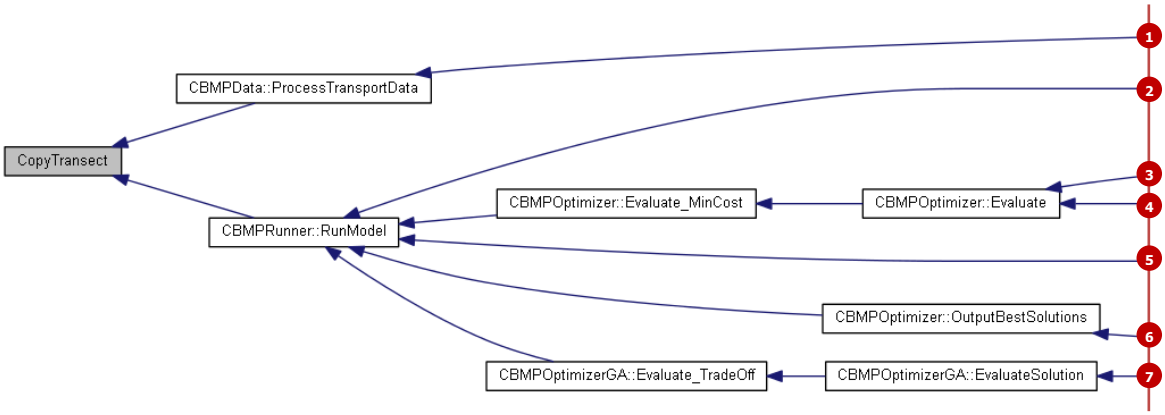
CopyTransect *(TTransect * Source, TTransect * Target)*

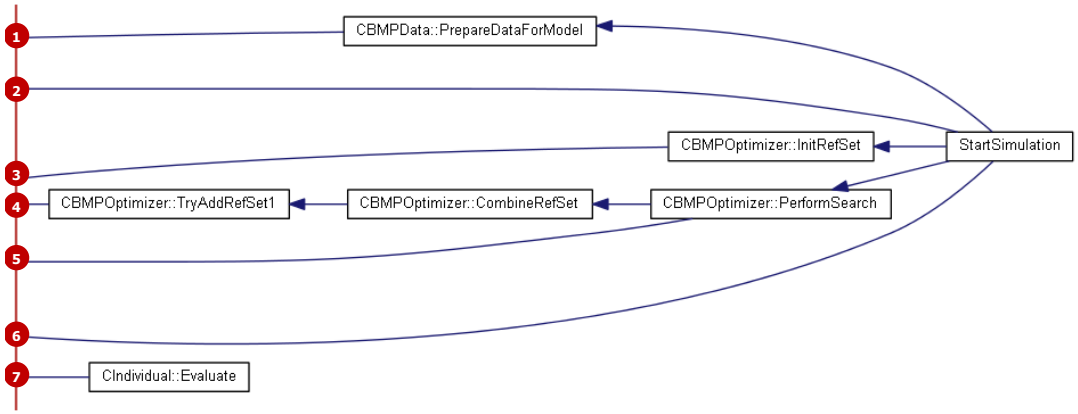
This function assigns the user-defined transect parameters to another transect object.

Parameters:

Source	The source data structure.
Target	The target data structure.

This is the caller graph for this function:





FindObIndexFromList *(CObList &* **list***, CObject ** **ob***)*

This function finds the object index from the list of objects.

Parameters:

in,out	*list*	If non-null, a pointer to the list of objects.
in,out	*ob*	If non-null, a pointer to the object.

Returns:

An index of the given object from the list of objects.

InitializeGAInfil *(int* **nNum***)*

This function initializes the Green-Ampt infiltration parameters.

Parameters:

nNum	The number of BMPs.

This is the caller graph for this function:

InitializeHortonInfil *(int* **nNum***)*

This function initializes the Horton infiltration parameters.

Parameter:

nNum	The number of BMPs.

This is the caller graph for this function:

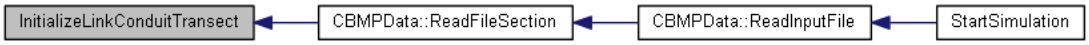
InitializeLinkConduitTransect (int nNum)

This function initializes the link, conduit, and transect parameters.

Parameter:

nNum	The number of objects.

This is the caller graph for this function:



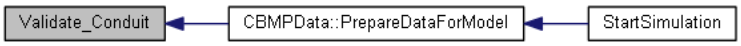
Validate_Conduit (int nNum)

This function validates the conduit parameters.

Parameter:

nNum	number of Conduits.

This is the caller graph for this function:



BMPData.h File Reference

#include <afxtempl.h>
#include <queue>

Classes

- struct POLLUTANT
- struct BMPCOST
- class CBMPData

Defines

- #define STRATEGY_SCATTER_SEARCH 1
- #define STRATEGY_GENETIC_ALGORITHM 2

Detailed Description

This is header file for the CBMPData class.

Define Documentation

*#define **STRATEGY_GENETIC_ALGORITHM** 2*

This is the definition for the Genetic Algorithm optimization method.

*#define **STRATEGY_SCATTER_SEARCH** 1*

This is the definition for the Scatter Search optimization method.

BMPOptimizer.cpp File Reference

```
#include "stdafx.h"
#include "Global.h"
#include "BMPSite.h"
#include "BMPData.h"
#include "BMPRunner.h"
#include "BMPOptimizer.h"
#include "ProgressWnd.h"
#include <math.h>
#include <float.h>
```

Detailed Description

This file implements the CBMPOptimizer class.

BMPOptimizer.h File Reference

Classes

- class CBMPOptimizer

Detailed Description

This is header file for the CBMPOptimizer class.

BMPOptimizerGA.cpp File Reference

```
#include "stdafx.h"
#include "Global.h"
#include "Individual.h"
#include "Population.h"
#include "BMPSite.h"
#include "BMPData.h"
#include "BMPRunner.h"
#include "BMPOptimizerGA.h"
#include <math.h>
#include <float.h>
```

Detailed Description

This file implements the CBMPOptimizerGA class.

BMPOptimizerGA.h File Reference

Classes

- class CBMPOptimizerGA

Detailed Description

This is header file for the CBMPOptimizerGA class.

BMPRunner.cpp File Reference

```
#include "stdafx.h"
#include "Global.h"
#include "LandUse.h"
#include "BMPSite.h"
#include "SiteLandUse.h"
#include "SitePointSource.h"
#include "Sediment.h"
#include "BMPData.h"
#include "BMPRunner.h"
#include <math.h>
#include <afxtempl.h>
#include "ProgressWnd.h"
#include "BMPOptimizer.h"
#include "Aquifer.h"
#include "Pump.h"
```

Global Functions

- void findLinkQual2 (int i, float tStep, double wAdded, double kDecay, double &c)
- float getHydRad (TXsect *xsect, float y)

Detailed Description

This file implements the CBMPRunner class.

Function Documentation

void ***findLinkQual2*** *(int* i*, float* tStep*, double* wAdded*, double* kDecay*, double &* c*)*

This function finds new quality in a conduit after the current time step.

Parameters:

	i	Zero-based index of the conduit.
	tStep	Routing time step (sec).
	wAdded	The added material (lb/ivl).
	kDecay	The first order decay rate (per sec).
in,out	*c*	The updated concentration (lb/ft^3).

float ***getHydRad*** *(TXsect ** xsect*, float* y*)*

This function gets the hydraulic radius for the conduit cross-section and water depth.

Parameters:

in,out	*xsect*	If non-null, the pointer to the conduit cross-section data structure.
	y	The float; water depth (ft).

Returns:

The hydraulic radius (ft).

BMPRunner.h File Reference

Classes

- class CBMPRunner

Defines

- #define RUN_INIT 0
- #define RUN_OPTIMIZE 1
- #define RUN_OUTPUT 2
- #define RUN_PREDEV 3
- #define RUN_POSTDEV 4
- #define TOTAL 1
- #define WEIR_ 2
- #define ORIFICE_CHANNEL 3
- #define UNDERDRAIN 4
- #define OPTION_NO_OPTIMIZATION 0
- #define OPTION_MIMIMIZE_COST 1
- #define OPTION_TRADE_OFF_CURVE 2
- #define AAFV -1
- #define PDF -2
- #define FEF -3
- #define AAL 1
- #define AAC 2
- #define MAC 3
- #define CALC_PERCENT 1
- #define CALC_SCALE 2
- #define CALC_VALUE 3
- #define SAND 1
- #define SILT 2
- #define CLAY 3
- #define TSS 4
- #define POUND2GRAM 453.5924
- #define LBpCFT2MGpL 16018.46
- #define CFS2CMS 0.0283
- #define CF2CM 0.0283
- #define FpS2MpS 0.3048
- #define FOOT2METER 0.3048
- #define fThreshold 1.0e-7

Detailed Description

This is header file for the CBMPRunner class.

Define Documentation

*#define **AAC** 2*

This is the definition for evaluation factor type: average annual concentration (mg/L).

*#define **AAFV** -1*

This is the definition for evaluation factor type: annual average flow volume (ft^3/yr).

*#define **AAL** 1*

This is the definition for evaluation factor type: annual average load (lb/yr).

*#define **CALC_PERCENT** 1*

This is the definition for calculating the percent value.

#define ***CALC_SCALE*** *2*

This is the definition for calculating the scale value.

#define ***CALC_VALUE*** *3*

This is the definition for calculating the absolute value.

#define ***CF2CM*** *0.0283*

This is the definition for converting units from ft^3 to m^3.

#define ***CFS2CMS*** *0.0283*

This is the definition for converting units from cfs to cms.

#define ***CLAY*** *3*

This is the definition for the sediment class: clay identifier.

#define ***FEF*** *-3*

This is the definition for the evaluation factor type: flow exceeding frequency (cfs).

#define ***FOOT2METER*** *0.3048*

This is the definition for converting units from feet to meters.

#define ***FpS2MpS*** *0.3048*

This is the definition for converting units from ft/s to m/s.

#define ***fThreshold*** *1.0e-7*

This is the definition for the flow threshold (cfs).

#define ***LBpCFT2MGpL*** *16018.46*

This is the definition for converting units from lbs/ft^3 to mg/L.

#define ***MAC*** *3*

This is the definition of the evaluation factor type: maximum days of average conc. (mg/L).

#define ***OPTION_MIMIMIZE_COST*** *1*

This is the definition of the BMP optimization option: mimimize cost.

#define ***OPTION_NO_OPTIMIZATION*** *0*

This is the definition of the BMP optimization option: no optimization.

#define ***OPTION_TRADE_OFF_CURVE*** *2*

This is the definition of the BMP optimization option: cost-effective curve.

#define ***ORIFICE_CHANNEL*** *3*

This is the definition of the BMP orifice/channel outflow flag.

#define ***PDF*** *-2*

This is the definition of the evaluation factor type: peak discharge flow (cfs).

#define ***POUND2GRAM*** *453.5924*

This is the definition of converting units from pounds to grams.

#define ***RUN_INIT*** *0*

This is the definition of the BMP simulation flag for the baseline scenario with existing BMPs.

#define ***RUN_OPTIMIZE*** *1*

This is the definition of the BMP simulation flag for the optimize scenario.

#define ***RUN_OUTPUT*** *2*

This is the definition of the BMP simulation flag for the best solution scenario.

#define ***RUN_POSTDEV*** *4*

This is the definition of the BMP simulation flag for the post-developed scenario.

#define ***RUN_PREDEV*** *3*

This is the definition of the BMP simulation flag for the pre-developed scenario.

#define ***SAND*** *1*

This is the definition of the sediment class flag: sand identifier.

#define ***SILT*** *2*

This is the definition of the sediment class flag: silt identifier.

#define ***TOTAL*** *1*

This is the definition of the BMP total outflow identifier.

#define ***TSS*** *4*

This is the definition of the sediment class flag: total sediment identifier.

#define ***UNDERDRAIN*** *4*

This is the definition of the BMP underdrain outflow identifier.

#define ***WEIR_*** *2*

This is the definition of the BMP weir outflow identifier.

BMPSite.cpp File Reference

#include "stdafx.h"
#include "BMPSite.h"
#include "StringToken.h"
#include "./swmm5/odesolve.h"
#include <math.h>

Global Functions

- void getDdDt2 (float t, float *d, float *dddt)
- void getDdDt3 (float t, float *d, float *dddt)

Detailed Description

This file implements the CBMPSite class.

Function Documentation

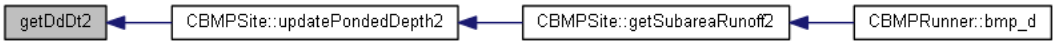
*getDdDt2 (float t, float * d, float * dddt)*

This function evaluates the derivative of stored depth with respect to time for the BUFFERSTRIP whose runoff is being computed.

Parameters:

	t	The current time (not used).
in,out	*d*	The stored depth (ft).
in,out	*dddt*	The derivative of d with respect to time.

This is the caller graph for this function:



*getDdDt3 (float t, float * d, float * dddt)*

This function evaluates the derivative of stored depth with respect to time for the AREABMP whose runoff is being computed.

Parameters:

	t	current time (not used).
in,out	*d*	The stored depth (ft).
in,out	*dddt*	The derivative of d with respect to time.

This is the caller graph for this function:

BMPSite.h File Reference

```
#include <queue>
#include "./SWMM5/headers.h"
```

Classes

- struct PUMP_CONTROL
- struct ADJUSTABLE_PARAM
- struct COST_PARAM
- struct EVALUATION_FACTOR
- struct HOLTAN_PARAM
- struct BMP_A
- struct BMP_B
- struct BMP_C
- struct BMP_D
- struct BMP_E
- struct SEDIMENT
- struct SAND
- struct SILTCLAY
- class TradeOffCurve
- class POLLUT_RAConc
- class CBMPSite
- struct DS_BMPSITE
- struct US_BMPSITE

Defines

- #define CLASS_A 1
- #define CLASS_B 2
- #define CLASS_C 3
- #define CLASS_D 4
- #define CLASS_E 5
- #define CLASS_X 100

Detailed Description

This is header file for the CBMPSite class.

Define Documentation

*#define **CLASS_A** 1*

This is the definition flag for BMP Class A. This BMP class represents the practices that capture upstream drainage at a specific location and can use a combination of detention, infiltration, evaporation, settling, and transformation to manage flow and remove pollutants.

*#define **CLASS_B** 2*

This is the definition flag for BMP Class B. This BMP class represents the channel type management practices such as swale that does not necessarily detain the flow but provides infiltration, evaporation, and settling to attenuate the flow and remove pollutants.

*#define **CLASS_C** 3*

This is the definition flag for BMP Class C. This BMP class represents the conveyance system such as a pipe or open channel.

*#define **CLASS_D** 4*

This is the definition flag for BMP Class D. This BMP class represents the linear type practices such as a buffer strip.

#define ***CLASS_E*** *5*

This is the definition flag for BMP Class E. This BMP class represents the nonstructural practices such as disconnected impervious areas routed to pervious land segment.

#define ***CLASS_X*** *100*

This is the definition flag for BMP Class X. This BMP class represents the outlet node such as a junction.

Global.cpp File Reference

```
#include "stdafx.h"
#include "Global.h"
#include "BMPSite.h"
#include "BMPData.h"
#include "BMPRunner.h"
#include "BMPOptimizer.h"
#include "BMPOptimizerGA.h"
#include "ProgressWnd.h"
#include "StringToken.h"
#include "./swmm5/swmm5.h"
#include "./swmm5/odesolve.h"
#include <math.h>
```

Global Functions

- double random_perc ()
- int random_int (int low, int high)
- double random_real (double low, double high)
- double random_real_with_inc (double low, double high, double inc)
- BOOL PASCAL EXPORT StartLandSimulation (char *strLandPreDevFilePath, char *strLandPostDevFilePath)
- BOOL PASCAL EXPORT StartSimulation (char *strInputFilePath, char *strBestPopRun, char *strRun_Mode)
- int LandSimulation (int landfg, char *strInputFilePath, CProgressWnd *pwndProgress)
- double pet_Hamon (double lat, double cts, double tavc, double day)

Detailed Description

This file implements the exported function for *SUSTAIN* DLL.

Function Documentation

int **LandSimulation** *(int* landfg*, char ** strInputFilePath*, CProgressWnd ** pwndProgress*)*

This function calls the SWMM for internal land simulation option in *SUSTAIN*.

Parameters:

landfg	The flag indicating the pre/post development scenario (landfg=0 for pre-development and landfg=1 for post-development).
strInputFilePath	The full path of the SWMM input text file created by *SUSTAIN* land simulation module.
pwndProgress	The pointer to the CProgressWnd class for the land simulation progress bar.

Returns:

The error code if the simulation fails otherwise 0.

This is the caller graph for this function:

double **pet_Hamon** *(double* lat*, double* cts*, double* tavc*, double* day*)*

This function calculates daily PET value based on the Hamon method.

Parameters:

lat	The latitude.
cts	A monthly variable coefficient.
tavc	The mean daily air temperature (degrees C).
day	The day of the year.

Returns:

The PET value for this day of the year (in./day).

int ***random_int*** *(int* low, *int* high*)*

This function generates a random integer between the low and high values including the bounds.

Parameters:

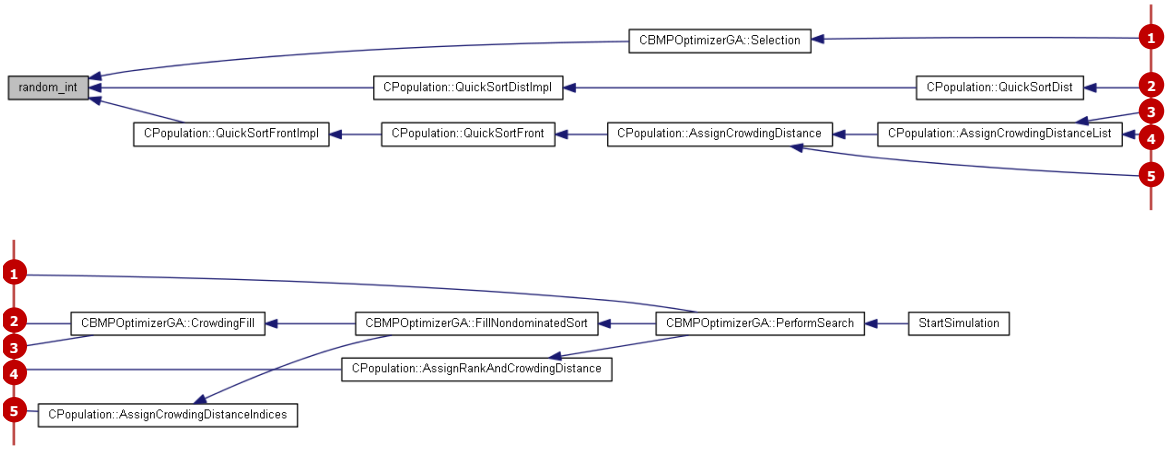
low	The lower bound of the random integer to be generated.
high	The upper bound of the random integer to be generated.

Returns:

The generated random integer.

This is the call graph for this function:

This is the caller graph for this function:



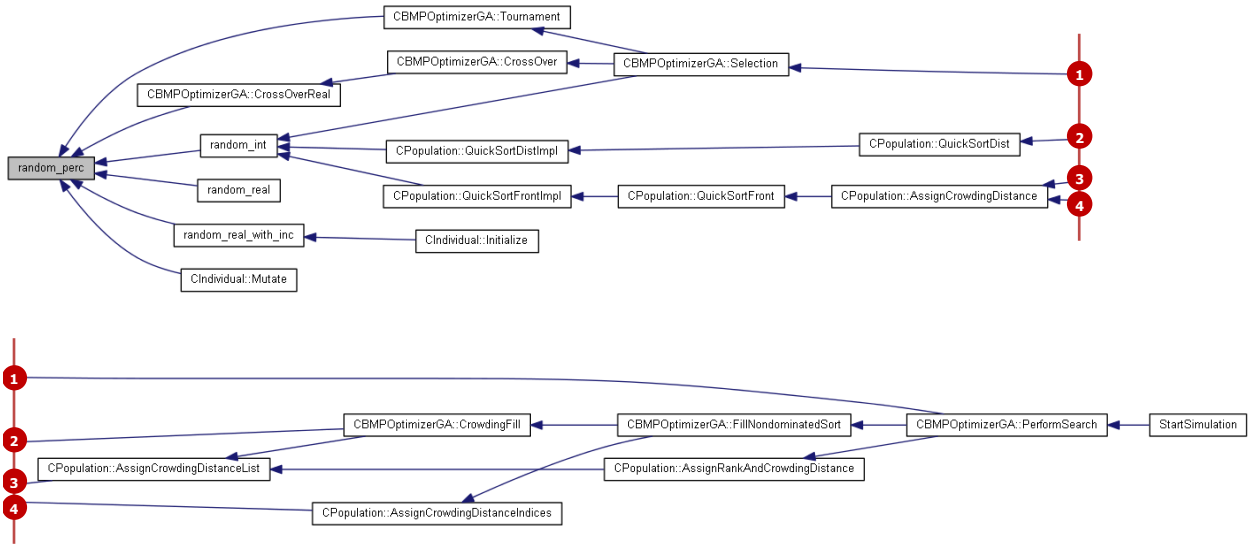
double ***random_perc*** *()*

This function generates a random number between 0.0 and 1.0.

Returns:

A random number that is between 0.0 and 1.0 with up to four digits.

This is the caller graph for this function:



double ***random_real*** *(double* low, *double* high*)*

This function generates a random real number between the user-specified low and high values including the bounds.

Parameters:

low	The lower bound of the random real number to be generated.
high	The upper bound of the random real number to be generated.

Returns:

The generated random real number.

This is the call graph for this function:

double ***random_real_with_inc*** *(double* low, *double* high, *double* inc*)*

This function generates a random real number between the user-specified low and high values with user-specified increment, including the bounds.

Parameters:

low	The lower bound of the random real number to be generated.
high	The upper bound of the random real number to be generated.
inc	The increment of the random number for the lower bound.

Returns:

The generated random real number.

This is the call graph for this function:

This is the caller graph for this function:

C BOOL PASCAL EXPORT ***StartLandSimulation*** *(char * strLandPreDevFilePath, char * strLandPostDevFilePath)*

This function is the DLL entry point for the *SUSTAIN* GIS interface to run the land simulation module.

Parameters:

strLandPreDevFilePath	The full path for the pre-development land simulation input file.
strLandPostDevFilePath	The full path for the post-development land simulation input file.

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:

C BOOL PASCAL EXPORT ***StartSimulation*** *(char * strInputFilePath, char * strBestPopRun, char * nRun_Mode)*

This function is the DLL entry point for the *SUSTAIN* GIS interface to run the BMP simulation module.

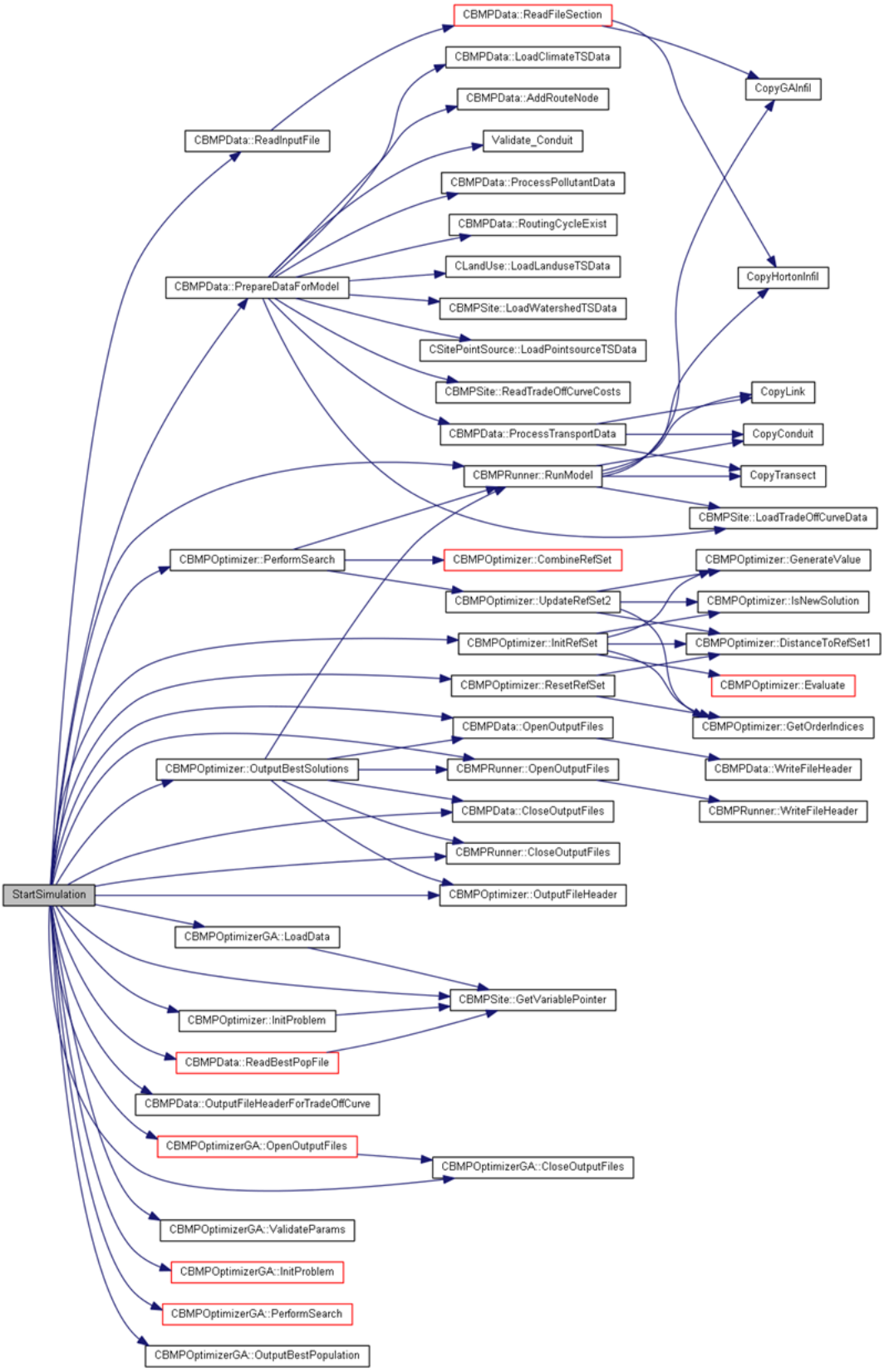
Parameters:

strInputFilePath	The full path for the BMP simulation input file.
strBestPopRun	The best population ID to run the optimized solution from the cost-effective curve.
nRun_Mode	A flag indicating the BMP simulation run mode (0=regular mode, 1=batch mode).

Returns:

True if it succeeds, false if it fails.

This is the call graph for this function:



Global.h File Reference

Classes

- struct SCATTER_SEARCH
- struct GA_PROBLEM

Defines

- #define INF 1.0e14
- #define EPS 1.0e-14
- #define E 2.71828182845905
- #define pi 3.14159265358979
- #define SUSTAIN_VERSION "Version 1.2 - November 30, 2011"

Detailed Description

This is the main header file for the SUSTAINOPT DLL. It defines the values for global constants, optimization data structure, and export functions for DLL entry points from the *SUSTAIN* GIS interfaces.

Define Documentation

*#define **E** 2.71828182845905*

This is the definition of the base of natural logarithm.

*#define **EPS** 1.0e-14*

This is the definition of the infinitely small value.

*#define **INF** 1.0e14*

This is the definition of the infinitely large value.

*#define **pi** 3.14159265358979*

This is the definition of the value of pi.

*#define **SUSTAIN_VERSION** "Version 1.2 - November 30, 2011"*

This is the definition of the software version.

Individual.cpp File Reference

```
#include "stdafx.h"
#include "Global.h"
#include "Individual.h"
#include "BMPOptimizerGA.h"
#include <math.h>
```

Functions

- double random_perc ()
- double random_real (double low, double high)
- double random_real_with_inc (double low, double high, double inc)

Detailed Description

This file implements the CIndividual class. The three external functions for random number or percentage generation are as previously defined in the Global.cpp file.

Function Documentation

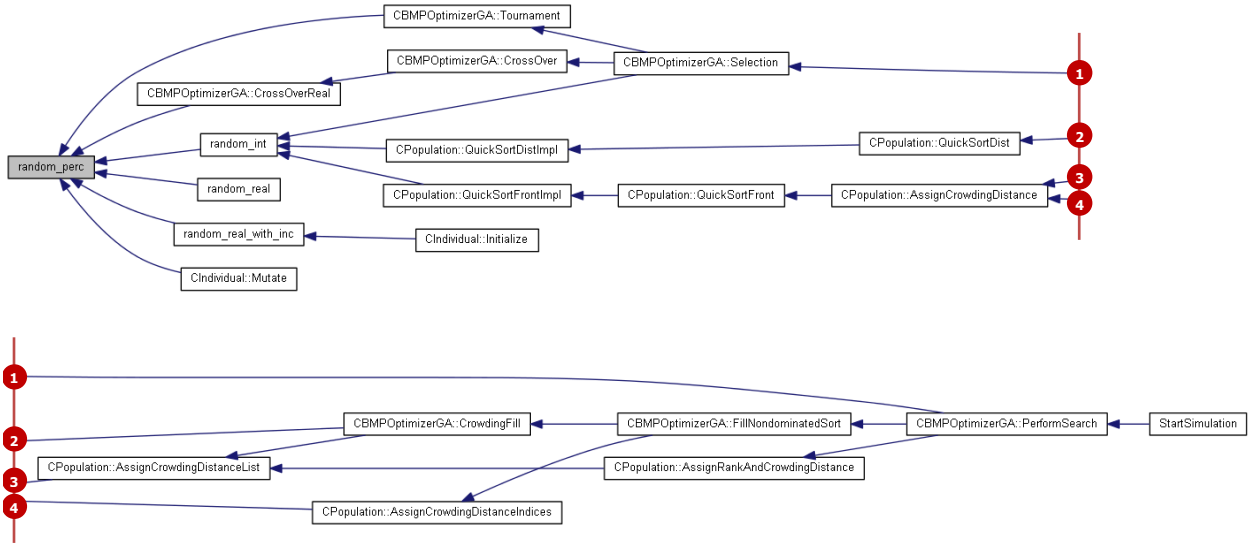
double ***random_perc*** *()*

This function generates a random number between 0.0 and 1.0.

Returns:

A random number that is between 0.0 and 1.0 with up to four digits.

This is the caller graph for this function:



double ***random_real*** *(double* low*, double* high*)*

This function generates a random real number between the user-specified low and high values including the bounds.

Parameters:

low	The lower bound of the random real number to be generated.
high	The upper bound of the random real number to be generated.

Returns:

The generated random real number.

This is the call graph for this function:

double ***random_real_with_inc*** *(double* low, *double* high, *double* inc*)*

This function generates a random real number between the user-specified low and high values with user-specified increment including the bounds.

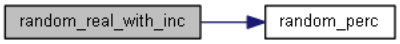
Parameters:

low	The lower bound of the random real number to be generated.
high	The upper bound of the random real number to be generated.
inc	The increment of the random number for the lower bound.

Returns:

The generated random real number.

This is the call graph for this function:



This is the caller graph for this function:

Individual.h File Reference

Classes

- class CIndividual

Detailed Description

This is header file for the CIndividual class.

LandUse.cpp File Reference

```
#include "stdafx.h"
#include "LandUse.h"
#include "StringToken.h"
```

Detailed Description

This file implements the CLandUse class.

LandUse.h File Reference

Classes

- class CLandUse

Detailed Description

This is header file for the CLandUse class.

Population.cpp File Reference

```
#include "stdafx.h"
#include "Global.h"
#include "Individual.h"
#include "Population.h"
#include <afxtempl.h>
#include <math.h>
#include <float.h>
```

Functions

- int random_int (int low, int high)

Detailed Description

This file implements the CPopulation class.

Function Documentation

int ***random_int*** *(int* low, *int* high*)*

This function generates a random integer between the low and high values including the bounds.

Parameters:

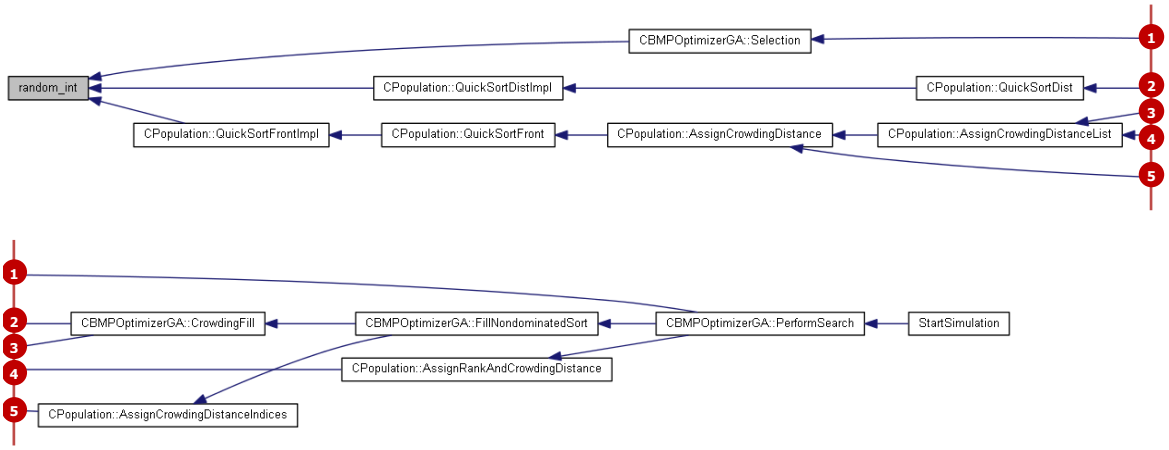
low	The lower bound of the random integer to be generated.
high	The upper bound of the random integer to be generated.

Returns:

The generated random integer.

This is the call graph for this function:

This is the caller graph for this function:



Population.h File Reference

Classes

- class CPopulation

Detailed Description

This is header file for the CPopulation class.

Pump.cpp File Reference

```
#include "stdafx.h"
#include "Pump.h"
#include <math.h>
```

Detailed Description

This file implements the CPUMP class.

Pump.h File Reference

Classes

- class CPUMP

Detailed Description

This is header file for the CPUMP class.

Resource.h File Reference

Detailed Description

This is Microsoft Visual C++ generated resource file.

Sediment.cpp File Reference

#include "stdafx.h"
#include <math.h>
#include <string.h>
#include "Sediment.h"

Global Functions

- float dayval (float mval1, float mval2, int day, int ndays)
- int detach (int crvfg, int csnofg, int mon, int nxtmon, int day, int ndays, float *coverm, float rain, float snocov, float delt60, float smpf, float krer, float jrer, float &cover, float &dets, float &det)
- int attach (float affix, float deltd, float &dets)
- int sosed1 (float runoff, float surs, float delt60, float kser, float jser, float kger, float jger, float &dets, float &sosed)
- int BDEXCH (double AVDEPM, double W, double TAU, double TAUCD, double TAUCS, double M, double VOL, double FRCSED, double *SUSP, double *BED, double *DEPSCR)
- int sandld (double isand, double vols, double srovol, double vol, double erovol, double ksand, double avvele, double expsnd, double rom, int sandfg, double db50e, double hrade, double slope, double tw, double wsande, double twide, double db50m, double fsl, double avdepe, double *sand, double *rsand, double *bdsand, double *depscr, double *rosand)

Detailed Description

This file implements the sediment simulation functions.

Function Documentation

int **attach** *(float* affix*, float* deltd*, float &* dets*)*

This function simulates attachment or compaction of detached sediment on the surface. If the previous day was dry, the calculation is done at the start of the day.

Parameters:

	affix	The fraction by which detached sediment storage decreases each day as a result of soil compaction.
	deltd	Number of days in the time step.
in,out	*dets*	The storage of detached sediment (tons/ac).

int **BDEXCH** *(double* AVDEPM*, double* W*, double* TAU*, double* TAUCD*, double* TAUCS*, double* M*, double* VOL*, double* FRCSED*, double ** SUSP*, double ** BED*, double ** DEPSCR*)*

This function simulates deposition and scour of cohesive sediment.

Parameters:

	AVDEPM	The average depth of water in meters .
	W	The settling velocity for cohesive sediment fraction (len/ivl).
	TAU	The shear stress (lb/ft^2).
	TAUCD	The critical shear stress for deposition (lb/ft^2).
	TAUCS	The critical shear stress for scour (lbs/ft^2).
	M	The erodibility coefficient for the sediment fraction.
	VOL	The volume of water in BMP (ft^3).
	FRCSED	The fraction of sediment in the bed.
in,out	*SUSP*	The suspended storage of sediment fraction.
in,out	*BED*	The storage of sediment fraction in bed.
in,out	*DEPSCR*	The deposition (positive) or scour (negative).

float **dayval** *(float* mval1*, float* mval2*, int* day*, int* ndays*)*

This function linearly interpolates a value for this day using the values for the start of the month and the next month.

Parameters:

mval1	The value at the start of this month.
mval2	The value at the start of the next month.
day	The day of this month.
ndays	The number of days in this month.

Returns:

The value for this day.

This is the caller graph for this function:

int **detach** *(int* crvfg*, int* csnofg*, int* mon*, int* nxtmon*, int* day*, int* ndays*, float ** coverm*, float* rain*, float* snocov*, float* delt60*, float* smpf*, float* krer*, float* jrer*, float &* cover*, float &* dets*, float &* det*)*

This function calculates the rate of soil detachment and updates the detached sediment storage.

Parameters:

	crvfg	If crvfg is 1, erosion-related cover may vary throughout the year (crvfg = 0 in *SUSTAIN*).
	csnofg	If csnofg is 1, the snow accumulation and melt is being considered (csnofg = 0 in *SUSTAIN*).
	mon	This month.
	nxtmon	Next month.
	day	This day.
	ndays	Number of days in this month.
	rain	The rainfall (in./timestep).
	snocov	The fraction of land use covered by snow pack (snocov = 0 in *SUSTAIN*).
	delt60	Number of hours in the simulation timestep.
	smpf	The supporting management practice factor used in sediment detach equation.
	krer	The detachment coefficient dependent on soil properties.
	jrer	The detachment exponent dependent on soil properties.
in,out	*coverm*	If non-null, monthly values of the cover parameter.
in,out	*cover*	The fraction of land surface that is shielded from rainfall erosion.
in,out	*dets*	The storage of detached sediment (tons/ac).
in,out	*det*	The sediment detached from the soil matrix by rainfall (tons/ac/interval).

This is the call graph for this function:

***int sandld** (double* **isand**, *double* **vols**, *double* **srovol**, *double* **vol**, *double* **erovol**, *double* **ksand**, *double* **avvele**, *double* **expsnd**, *double* **rom**, *int* **sandfg**, *double* **db50e**, *double* **hrade**, *double* **slope**, *double* **tw**, *double* **wsande**, *double* **twide**, *double* **db50m**, *double* **fsl**, *double* **avdepe**, *double ** **sand**, *double ** **rsand**, *double ** **bdsand**, *double ** **depscr**, *double ** **rosand**)*

This function simulates the behavior of sand.

Parameters:

	isand	The inflow of sand over the timestep.
	vols	The volume of water at the start of timestep (ft^3).
	srovol	The outflow volume component based on the start of the interval (ft^3/timestep).
	vol	The volume of water at the end of the timestep (ft^3).
	erovol	The outflow volume component based on the end of the interval (ft^3/timestep).
	ksand	The coefficient in the sandload power function formula.
	avvele	The average flow velocity (ft/s).
	expsnd	The exponent in the sandload power function formula.
	rom	The total rate of outflow of water (m^3/sec).
	sandfg	It indicates the method that will be used for sandload simulation (*SUSTAIN* supports user-specified power function method).
	db50e	The median diameter of bed material (ft).
	hrade	The hydraulic radius (ft).
	slope	The longitudinal slope (ft/ft).
	tw	The water temperature (degree C).
	wsande	The fall velocity of sand (ft/sec).
	twide	The width of reach (ft).
	db50m	The median diameter of bed material (m).
	fsl	The fine sediment load concentration.
	avdepe	The average water depth (ft).
in,out	*sand*	The concentration of sand in suspension (mg/L).
in,out	*rsand*	The sand material in suspension (g).
in,out	*bdsand*	The sand material available in the bed.
in,out	*depscr*	The deposition (positive) or scour (negative).
in,out	*rosand*	The total amount of sand leaving the reach during the timestep (g).

***int sosed1** (float* **runoff**, *float* **surs**, *float* **delt60**, *float* **kser**, *float* **jser**, *float* **kger**, *float* **jger**, *float &* **dets**, *float &* **sosed**)*

This function calculates the washoff from both detached surface sediment and soil matrix by surface runoff.

Parameters:

	runoff	The surface runoff (in./timestep).
	surs	The surface water storage (in.).
	delt60	Number of hours in the simulation timestep.
	kser	The coefficient in the detached sediment washoff equation.
	jser	The exponent in the detached sediment washoff equation.
	kger	The coefficient in the matrix soil scour equation.
	jger	The exponent in the matrix soil scour equation.
in,out	*dets*	The storage of detached sediment (tons/ac).
in,out	*sosed*	Total removal of soil and sediment (tons/ac/timestep).

Sediment.h File Reference

Detailed Description

This is header file for the sediment simulation functions.

SiteLandUse.cpp File Reference

#include "stdafx.h"
#include "LandUse.h"
#include "BMPSite.h"
#include "SiteLandUse.h"

Detailed Description

This file implements the CSiteLandUse class.

SiteLandUse.h File Reference

Classes

- class CSiteLandUse

Detailed Description

This is header file for the CSiteLandUse class.

SitePointSource.cpp File Reference

```
#include "stdafx.h"
#include "BMPSite.h"
#include "StringToken.h"
#include "SitePointSource.h"
```

Detailed Description

This file implements the CSitePointSource class.

SitePointSource.h File Reference

Classes

- class CSitePointSource

Detailed Description

This is header file for the CSitePointSource class.

StdAfx.cpp File Reference

#include "stdafx.h"

Detailed Description

This file includes the stdafx header file.

StdAfx.h File Reference

```
#include <afxwin.h>
#include <afxext.h>
#include <afxole.h>
#include <afxodlgs.h>
#include <afxdisp.h>
#include <afxdb.h>
#include <afxdao.h>
#include <afxdtctl.h>
#include <afxcmn.h>
```

Detailed Description

This file includes the standard or project-specific include files that are used frequently.

SUSTAIN.cpp File Reference

#include "stdafx.h"
#include "SUSTAIN.h"

Detailed Description

This file implements the CSUSTAINApp class.

SUSTAIN.h File Reference

#include "resource.h"

Detailed Description

This is header file for the CSUSTAINApp class.

References

Bicknell, B.R., J.C. Imhoff, J.L. Kittle Jr., T.H. Jobes, and A.S. Donigian Jr. 2001. *Hydrological Simulation Program—FORTRAN, Version 12, User's Manual*. U.S. Environmental Protection Agency, National Exposure Research Laboratory, Athens, GA., in cooperation with U.S. Geological Survey, Water Resources Division, Reston, VA.

Deb, K., A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multi-objective genetic algorithm: NSGAII. *IEEE Transactions on Evolutionary Computation* 6(2):182–197.

Glover, F., M. Laguna, and R. Marti. 2000. Fundamentals of Scatter Search and Path Relinking. *Control and Cybernetics* 29(3):653–684.

Glover, F. 1977. Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences* 8(7):156–166.

Hamon, W.R. 1961. Estimating Potential Evapotranspiration. *Journal of the Hydraulic Division*. *Proceedings* of the *American Society* of *Civil Engineers* 87:107–120.

Huber, W.C., and R.E. Dickinson. 1988. *Storm Water Management Model Version 4, User's Manual*. EPA 600/388/001a (NTIS PB88-236641/AS). U.S. Environmental Protection Agency, Athens, GA.

Kadlec, R.H., and R.L. Knight. 1996. *Treatment Wetlands*. CRC Press, Lewis Publishers, Boca Raton, FL.

Laguna, M., and R. Marti. 2002. *The OptQuest Callable Library to Appear in Optimization Software Class Libraries*. eds. S. Voss and D.L. Woodruff, pp. 193–218. Kluwer Academic Publishers, Boston, MA.

Linsley, R.K., J.B. Franzini, D.L. Freyberg, and G. Tchobanoglous. 1992. *Water-Resources Engineering*. 4th ed. McGraw-Hill, New York, NY.

Maidment, D.R., ed. 1993. *Handbook of Hydrology*. McGraw-Hill, New York, NY.

Partheniades, E. 1962. *A Study of Erosion and Deposition of Cohesive Soils in Sult Water*. Ph.D. dissertation. University of California, Berkeley, CA.

Persson, J., N.L.G. Sommes, and T.H.F. Wong. 1999. Hydraulics efficiency of constructed wetlands and ponds. *Water Science and Technology* 40(3):291–300.

Rossman, L.A. 2005. *Stormwater Management Model User's Manual, Version 5.0*. EPA/600/R-05/040. U.S. Environmental Protection Agency, Water Supply and Water Resources Division, National Risk Management Research Laboratory, Cincinnati, OH.

USBR (U.S. Bureau of Reclamation). 2001. *Water Measurement Manual*. U.S. Department of the Interior, Bureau of Reclamation, Washington, DC.

USEPA (U.S. Environmental Protection Agency). 1998. *Estimation of Infiltration Rate in the Vadose Zone: Compilation of Simple Mathematical Models Volume I*. U.S. Environmental Protection Agency, Washington, DC.

USEPA (U.S. Environmental Protection Agency). 2009. *SUSTAIN - A Framework for Placement of Best Management Practices in UrbanWatersheds to Protect Water Quality*. EPA/600/R-09/095. U.S. Environmental Protection Agency, Office of Research and Development, Cincinnati, OH.

Wong, T.H.F., H.P. Duncan, T.D. Fletcher, and G.A. Jenkins. 2001. A unified approach to modelling urban stormwater treatment. In *Proceedings of the Second South Pacific Stormwater Conference.* June 27–29, 2001, Auckland, New Zealand.

Wong, T.H.F., and P.F. Breen. 2002. Recent advances in Australian practice on the use of constructed wetlands for stormwater treatment. In *Global Solutions for Urban Drainage, Proceedings of the Ninth International Conference on Urban Drainage*, eds. E.W. Strecker and W.C. Huber. American Society of Civil Engineers, September 2002, Portland, OR. CD-ROM.

Zhen, X., and S.L. Yu. 2004. Optimal location and sizing of stormwater basins at watershed scale. *Journal of Water Resources Planning and Management* 130(4):339–347