An “IRF Interface” Example

Scott D. Peckham
CSDMS Integration Facility
University of Colorado

Joint Meeting of the CSDMS Terrestrial and Coastal Working Groups,
Boulder, Colorado
October 26, 2009

The Basic "IRF" Interface

In the context of componentized software, **an *interface* is a named set of member functions** that provide a caller with access to its capabilities. (That is, names and data types of all arguments & return values are completely specified.

A basic "IRF interface" is something that virtually all model coupling efforts have in common (e.g. ESMF, OMS and OpenMI). IRF stands for "Initialize, Run_Step, Finalize". **We want contributors to provide this interface.**

Run_Model()
    Initialize()
    while not(DONE): **Update**()
    Finalize()

Initialize() => Open files, read/compute initial values, allocate memory, etc.
Run_step() or Update() => Update all of the computed values (one step forward)
    Can call other functions, e.g. Update_Q, Update_v.
Finalize() => Close files, print report, free memory.

water_tank.py – /Users/scott/Desktop/Meetings/Boulder_TWG_Oct_2009/Water_Tank_Python/w...

```
# July 5-7, 2008
# Modified: July 23, 2008  (v = sqrt(2 * g * d))
# S. Peckham

from numpy import zeros, random, double
from math import pi, sqrt

from input_file import *
# from rainfall import *

#---------------------------------------------------------------
def unit_test():

    tank_model = water_tank()
    tank_model.run_model()

#   unit_test()
#---------------------------------------------------------------
#  Simulate water depth in a water tank with an open top.
#  Could instantiate with "tank_model = water_tank()"
#  Then calls are: tank_model.initialize(),
#                  tank_model.update(), etc.
#---------------------------------------------------------------
class water_tank:

    #-----------------------------------------------------------
    def run_model(self):

        self.initialize()

        for k in xrange( self.n_steps ):
            self.update()

        self.finalize()

    #   run_model()
    #-----------------------------------------------------------
    def initialize(self):

        self.g = 9.81    # [m/s^2]
        self.tank_data_filename  = 'tank_data.txt'

        #-------------------------------------
```

Ln: 1 Col: 0

water_tank.py - /Users/scott/Desktop/Meetings/Boulder_TWG_Oct_2009/Water_Tank_Python/wat...

```
    #-------------------------------------------------------------
    def initialize(self):

        self.g = 9.81    # [m/s^2]
        self.tank_data_filename  = 'tank_data.txt'

        #---------------------------------------
        # Read tank settings from "tank_file"
        #---------------------------------------
        self.read_tank_data()
        self.volume     = self.depth * self.top_area  #[m^3]
        self.out_area   = pi * self.out_radius**2.0
        self.print_tank_data()

        #--------------------------------
        # Open "rain_file" to read data
        #--------------------------------
        self.rain_file.open()

    #   initialize()
    #-------------------------------------------------------------
    def update(self):

        #-----------------------------------
        # Read next rainfall data entry...
        #-----------------------------------
        self.update_rain()
        rainrate_mps = self.rainrate / (3600.0 * 1000.0)
        Q_in = rainrate_mps * self.top_area   # [m^3/s]

        if (self.depth > 0):
            Q_out = self.velocity * self.out_area  # [m^3/s]
        else:
            Q_out = 0.0
        dVol = (Q_in - Q_out) * self.dt
        self.volume = max(self.volume + dVol, 0.0)
        self.depth  = (self.volume / self.top_area)
        self.velocity = sqrt(2.0 * self.g * self.depth)
        print 'depth =', self.depth, '[meters]'

        # Write new depth to an output file ?

    #   update()
    #-------------------------------------------------------------
    def get_value(self, value_code, time):
```

Ln: 45 Col: 55

water_tank.py - /Users/scott/Desktop/Meetings/Boulder_TWG_Oct_2009/Water_Tank_Python/wat...

```
    #-------------------------------------------------------------
    def get_value(self, value_code, time):

        #-----------------------------------------------------
        # Call update() method as many times as necessary
        # in order to compute the requested value at the
        # requested time.  Note that we do not override the
        # value of n_steps from tank_data_file.
        #-----------------------------------------------------
        n_steps = int(time / self.dt)
        for k in xrange( n_steps ):
            self.update()

        if (value_code == 0): return self.depth
        if (value_code == 1): return self.velocity
        if (value_code == 2): return self.volume
        if (value_code == 3): return self.rainrate
        if (value_code == 4): return self.duration

    #   get_value()
    #-------------------------------------------------------------
    def finalize(self):

        print ' '
        print 'Finished with water tank simulation.'
        print 'Final depth =', self.depth
        print ' '
        self.rain_file.close()

        # Close an output file ?

    #   finalize()
    #-------------------------------------------------------------
    def update_rain(self):

        if (self.n_steps <= self.rain_file.n_lines):
            record = self.rain_file.read_record()
            self.rainrate = record[0]
            self.duration = record[1]
        else: self.rainrate = 0.0

    #   update_rain()
    #-------------------------------------------------------------
    def read_tank_data(self):

        tank_file = input_file(self.tank_data_filename)
```

Ln: 97 Col: 0

*water_tank.py - /Users/scott/Desktop/Meetings/Boulder_TWG_Oct_2009/Water_Tank_Python/w...

```
    #-------------------------------------------------------------
    def read_tank_data(self):

        tank_file = input_file(self.tank_data_filename)
        # What if tank_file doesn't exist ?
        #-----------------------------------
        tank_file.open()
        self.dt          = tank_file.read_value()
        self.n_steps     = tank_file.read_value(dtype='integer')
        self.init_depth = tank_file.read_value()
        self.top_area    = tank_file.read_value()
        self.out_radius = tank_file.read_value()
        self.rain_data_filename = tank_file.read_value(dtype='string')
        tank_file.close()
        # These variables are computed from others
        self.depth       = self.init_depth.copy()
        self.velocity    = sqrt(2 * self.g * self.depth)
        self.volume      = self.depth * self.top_area  #[m^3]
        self.out_area    = pi * self.out_radius**2.0
        # Use "input_file" class to create rain_file object
        self.rain_file = input_file(self.rain_data_filename)

    #   read_tank_data()
    #-------------------------------------------------------------
    def print_tank_data(self):

        print 'dt          =', self.dt
        print 'n_steps     =', self.n_steps
        print 'init_depth =', self.init_depth
        print 'top_area    =', self.top_area
        print 'out_radius =', self.out_radius
        print 'depth       =', self.depth
        print 'velocity    =', self.velocity
        print 'volume      =', self.volume
        print 'out_area    =', self.out_area
        print 'rain_file   =', self.rain_data_filename
        print ' '

    #   print_tank_data()
    #-------------------------------------------------------------
```

Ln: 150 Col: 0