## Thermostat Measurement Bias

### Description

def description

return "Drift of the thermostat temperature sensor over time can lead to increased energy use and/or reduced occupant comfort. This measure simulates a biased thermostat by modifying the Schedule:Compact object in EnergyPlus assigned to heating and cooling set points. The fault intensity (F) for this fault is defined as the thermostat measurement bias (K). A positive number means that the sensor is reading a temperature higher than the true temperature."

end

### Modeler Description

def modeler\_description

return "Seven user inputs are required and, based on these user inputs, the original (non-faulted) heating and cooling set point schedules in the building model will be replaced with a biased temperature set point by the equation below. If the reading of the thermostat is biased with +1oC, the actual space temperature should be maintained 1oC lower than the reading. Thus, the set point for the space is corrected by subtracting the original set point from the biased level. T\_(stpt,heat,F)=T\_(stpt,heat)-F / T\_(stpt,cool,F)=T\_(stpt,cool)-F. The time required for the fault to reach the full level is only required when user wants to model dynamic fault evolution. If dynamic fault evolution is not necessary for the user, it can be defined as zero and the fault intensity will be imposed as a step function with user defined value. However, by defining the time required for the fault to reach the full level, fault starting month/date/time and fault ending month/date/time, the adjustment factor AF is calculated at each time step starting from the starting month/date/time to gradually impose fault intensity based on the user specified time frame. AF is calculated as follows, AF\_current = AF\_previous + dt/tau where AF\_current is the adjustment factor calculated based on the previously calculated adjustment factor (AF\_previous), simulation timestep (dt) and the time required for the fault to reach the full level (tau)."

end

### Measure Type

OpenStudio Measure

**Taxonomy**

HVAC.HVAC Controls

### Arguments

def arguments(model)

args = OpenStudio::Ruleset::OSArgumentVector.new

# make choice argument for thermal zone

zone\_handles, zone\_display\_names = pass\_zone(model, $allzonechoices)

zone = OpenStudio::Ruleset::OSArgument.makeChoiceArgument(

'zone', zone\_display\_names, zone\_display\_names, true

)

zone.setDefaultValue("\* All Zones \*")

zone.setDisplayName("Zone. Choose #{$allzonechoices} if you want to impose the fault in all zones")

args << zone

months = OpenStudio::StringVector.new

months << "January"

months << "February"

months << "March"

months << "April"

months << "May"

months << "June"

months << "July"

months << "August"

months << "September"

months << "October"

months << "November"

months << "December"

# heating season setpoint bias

bias\_level = OpenStudio::Ruleset::OSArgument::makeDoubleArgument("bias\_level", false)

bias\_level.setDisplayName("Enter the constant setpoint bias level [K] [0=Non faulted case]")

bias\_level.setDefaultValue(0)

args << bias\_level

#Parameters for transient fault modeling

#make a double argument for the time required for fault to reach full level

time\_constant = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('time\_constant', false)

time\_constant.setDisplayName('Enter the time required for fault to reach full level [hr]')

time\_constant.setDefaultValue(0) #default is zero

args << time\_constant

#make a double argument for the start month

start\_month = OpenStudio::Ruleset::OSArgument::makeChoiceArgument("start\_month", months, true)

start\_month.setDisplayName("Fault active start month")

start\_month.setDefaultValue("January")

args << start\_month

#make a double argument for the start date

start\_date = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('start\_date', false)

start\_date.setDisplayName('Enter the date (1-28/30/31) when the fault starts to occur')

start\_date.setDefaultValue(1) #default is 1st day of the month

args << start\_date

#make a double argument for the end month

end\_month = OpenStudio::Ruleset::OSArgument::makeChoiceArgument("end\_month", months, true)

end\_month.setDisplayName("Fault active end month")

end\_month.setDefaultValue("December")

args << end\_month

#make a double argument for the end date

end\_date = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('end\_date', false)

end\_date.setDisplayName('Enter the date (1-28/30/31) when the fault ends')

end\_date.setDefaultValue(31) #default is last day of the month

args << end\_date

return args

end

### Initial Condition

#Initial heating and cooling setpoints affected by the fault.

runner.registerInitialCondition("Initial heating setpoints in affected zones range from #{setpoint\_values[:init\_htg\_min].min.round(1)} C to #{setpoint\_values[:init\_htg\_max].max.round(1)} C. Initial cooling setpoints in affected zones range from #{setpoint\_values[:init\_clg\_min].min.round(1)} C to #{setpoint\_values[:init\_clg\_max].max.round(1)} C.")

### Final Condition

#Final heating and cooling setpoints affected by the fault.

runner.registerFinalCondition("Final heating setpoints in affected zones range from #{setpoint\_values[:final\_htg\_min].min.round(1)} C to #{setpoint\_values[:final\_htg\_max].max.round(1)} C. Final cooling setpoints in affected zones range from #{setpoint\_values[:final\_clg\_min].min.round(1)} C to #{setpoint\_values[:final\_clg\_max].max.round(1)} C.")

### Not Applicable

#When baseline model does not have proper schedule in selected zone,

runner.registerAsNotApplicable("No changes made, selected zones may not have had setpoint schedules, or schedules may not have been ScheduleRulesets.")

#When fault intensity is defined as zero,

runner.registerAsNotApplicable("No changes made thermostat bias of 0.0 requested.")

### Warning

#When thermostat is not defined in the selected zone,

runner.registerWarning("Cannot find existing thermostat for thermal zone '#{thermalzone.name}'. No changes made in this zone.")

#When heating schedule is not defined in the selected zone,

runner.registerWarning("Skipping #{thermalzone.name} because it is either missing heating setpoint schedule or the schedule is not ScheduleRulesets.")

#When cooling schedule is not defined in the selected zone,

runner.registerWarning("Skipping #{thermalzone.name} because it is either missing cooling setpoint schedule or the schedule is not ScheduleRulesets.")

### Error

#When user defined starting month is behind the ending month,

runner.registerError("Invalid fault start/end month combination.")

### Information

* Works with Schedule Ruleset.

### Code Outline

* Define arguments (zone where fault occurs, fault starting month, fault ending month, fault level in constant value).
* Check fault active months when fault is being imposed based on fault starting month and ending month.
* If time constant for fault evolution is defined as zero,
  + Apply fault for selected thermal zone(s).
    - Read heating and cooling thermostat schedule defined in the zone.
    - Gather thermostat setpoint information (average, min, max).
    - Alter heating thermostat setpoint based on biased level.
      * Copy original schedules.
      * Store setpoint values.
      * Empty schedules.
      * Insert biased setpoint values based on stored setpoint values.
    - Alter cooling thermostat setpoint based on biased level.
      * Copy original schedules.
      * Store setpoint values.
      * Empty schedules.
      * Insert biased setpoint values based on stored setpoint values.
    - Assign modified heating setpoints to thermostat.
    - Assign modified cooling setpoints to thermostat.
    - Assign thermostat to selected zone.
    - Gather modified thermostat setpoint information (average, min, max).
* If time constant for fault evolution is defined other than zero,
  + Append EMS code that calculates the adjustment factor (AF) and overwrites original heating and cooling schedules to selected zone.

### Tests

* Test different sets of input arguments (starting/ending month, biased level)
* Test invalid user argument values to make sure measure fails gracefully.