## HVAC Setback Error: Delayed Onset

### Description

def description

return "Thermostat schedules are employed to raise set points for cooling and lower set points for heating at night, to switch fan operation from being continuously on during occupied times to being coupled to cooling or heating demands at other times, and to close ventilation dampers during unoccupied periods. Faults can occur due to malfunctioning, unprogrammed, or incorrectly programmed or scheduled thermostats, leading to increased energy consumption and/or compromised comfort and air quality. This measure simulates the effect of overnight HVAC setback being delayed until unoccupied hours by modifying the Schedule:Compact object in EnergyPlus assigned to thermostat set point schedules. The fault intensity (F) for this fault is defined as the delay in onset of overnight HVAC setback (in hours)"

end

### Modeler Description

def modeler\_description

return "Five different user inputs are required to simulate the fault. The measure detects the original (non-faulted) thermostat schedule applied in EnergyPlus automatically, and adjusts the evening schedule based on user inputs. Note that this measure only works for buildings that become unoccupied before midnight. To use this Measure, choose the Zone that is faulted, and the period of time when you want the fault to occur. You should also enter the number of hours that the extension sustains. The measure will detect the thermostat schedule of the automatically, and adjust the evening schedule to the daytime schedule. Note that this measure only works for buildings close before midnight. You also need to choose one day in a week (Monday, Tuesday, .....) to simulate weekly fault occurence."

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(zone\_display\_names[0])

zone.setDisplayName('Zone')

args << zone

osmonths = OpenStudio::StringVector.new

$months.each do |month|

osmonths << month

end

start\_month = OpenStudio::Ruleset::OSArgument.makeChoiceArgument(

'start\_month', osmonths, true

)

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

start\_month.setDefaultValue($months[0])

args << start\_month

end\_month = OpenStudio::Ruleset::OSArgument.makeChoiceArgument(

'end\_month', osmonths, true

)

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

end\_month.setDefaultValue($months[11])

args << end\_month

osdaysofweeks = OpenStudio::StringVector.new

$dayofweeks.each do |day|

osdaysofweeks << day

end

osdaysofweeks << $all\_days

osdaysofweeks << $weekdaysonly

osdaysofweeks << $weekendonly

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

'dayofweek', osdaysofweeks, true

)

dayofweek.setDisplayName('Day of the week')

dayofweek.setDefaultValue($all\_days)

args << dayofweek

ext\_hr = OpenStudio::Ruleset::OSArgument.makeDoubleArgument('ext\_hr', true)

ext\_hr.setDisplayName(

'Number of operating hours extended to the evening.'

)

ext\_hr.setDefaultValue(1)

args << ext\_hr

return args

# note: the Assignment Branch Condition size is left higher than the

# recommended minimum by Rubocop because the argument definition

# functions are left in measure.rb to create json files automatically

end

### Initial Condition

#Initial heating and cooling setpoints affected by the fault.

runner.registerInitialCondition("Initial heating setpoints in affected zones range from #{setpoint\_values[:initial\_htg\_min].min.round(1)} C to #{setpoint\_values[:initial\_htg\_max].max.round(1)} C. Initial cooling setpoints in affected zones range from #{setpoint\_values[:initial\_clg\_min].min.round(1)} C to #{setpoint\_values[:initial\_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 fault measure is not applicable,

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

#When extended hours is set to zero,  
runner.registerAsNotApplicable('Zero hour extension in Measure ' \

'ExtendEveningThermostatSetpointWeek. ' \

'Exiting......')

### Warning

n/a

### Error

n/a

### Information

* Following measures share the same functions.
  + HVACSetbackErrorDelayedOnset
  + HVACSetbackErrorEarlyTermination
  + HVACSetbackErrorNoOvernightSetback
* Works with Schedule Ruleset.

### Code Outline

* Define arguments (zone where fault occurs, fault starting month, fault ending month, day of week when fault occurs, fault level in constant value).
* Check currently applied schedules (thermostat, heating or cooling).
* Gather setpoint values from those schedules (minimum and maximum).
* Create faulted schedule based on input arguments reflecting delayed onset of HVAC setback.
  + Create faulted schedule according to input arguments... addnewscheduleruleset\_ext\_hr
    - Create default schedule... createnewdefaultdayofweekrule\_ext\_hr
      * Create new schedule based on old schedule but with user defined fault starting month and ending month... createnewruleandcopy
        + Copy times and values from current schedule... copydayscheduletimesandvalues
        + Set fault starting date and ending date... Setcommoninformation
      * Change schedule type only applied to certain day of week... Changedayofweek
        + Apply schedule to all days in a week... applyallday
        + or Apply schedule to specific day in a week... applydayofweek
      * Propagate faulted schedule throughout the simulation period... propagateeveningchangeovervaluewithextrainfo\_ext\_hr
        + Find building opening time or closing time... findchangetime
        + Create schedule according to faulted time period... newtimesandvaluestosceduleday\_ext\_hr

Returns faulted time object according to faulted time period... shifttimevector

Updates faulted hours and minutes according to extended time... newhrandmin

Corrects time format within 24 hours... midnightadjust

Corrects hours and minutes to correct format... roundclock

* + - Create new priority schedule... createnewpriroityrules\_ext\_hr
      * Create new schedule based on old schedule but with user defined fault starting month and ending month... Createnewruleandcopy
        + Copy times and values from current schedule... copydayscheduletimesandvalues
        + Set fault starting date and ending date... Setcommoninformation
      * Compare and change the schedule according to faulted period... compareandchangedayofweek
        + Apply schedule to all days in a week... applyallday
        + Change schedule type only applied to certain day of week... changedayofweek

Apply schedule to all days in a week... applyallday

or Apply schedule to specific day in a week... Applydayofweek

* + - * Propagate faulted schedule throughout the simulation period... propagateeveningchangeovervalue\_ext\_hr
        + Find building opening time or closing time... findchangetime
        + Replace time and values in a schedule according to faulted period... newtimesandvaluestosceduleday
  + Add new heating and cooling setpoint schedules in DualSetpoint object... Addnewsetpointschedules
* Assign modified (or faulted) heating and cooling setpoint schedules to assigned thermostat.

### Tests

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