## Lighting Setback Error: Delayed Onset

### Description

def description

return ‘Lighting should be turned off or at least reduced during unoccupied hours. However, some commissioning studies have found noticeable lighting energy use at night either because lighting schedules are improperly configured or occupants forget to turn off lights when vacating a building (Haasl, Stum, and Arney 1996; Kahn, Potter, and Haasl 2002). This measure simulates the effect of lighting setback being delayed until unoccupied hours by modifying the Schedule:Compact object in EnergyPlus assigned to lighting schedules. The fault intensity (F) for this fault is defined as the delay in the onset of overnight lighting setback (in hours), which is specified as one of the inputs.'

end

### Modeler Description

def modeler\_description

return ‘Five different user inputs are required; zone where the fault occurs, starting month of the faulted operation, ending month of the faulted operation, day of the week when the fault occurs, delayed time in hours. The measure detects the original (non-faulted) lighting schedule applied in EnergyPlus automatically, and adjusts the schedule based on user inputs.'

end

### Measure Type

OpenStudio Measure

**Taxonomy**

Electric Lighting.Electric Lighting 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. Choose #{$allzonechoices} if you want to impose the fault in all zones")

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 lighting schedule fraction affected by the fault.

runner.registerInitialCondition("Initial lighting profile in affected zones range from #{setpoint\_values[:initial\_ltg\_min].min.round(1)} to #{setpoint\_values[:initial\_ltg\_max].max.round(1)}")

### Final Condition

#Final lighting schedule fraction affected by the fault.

runner.registerFinalCondition("Final lighting profile in affected zones range from #{setpoint\_values[:final\_ltg\_min].min.round(1)} to #{setpoint\_values[:final\_ltg\_max].max.round(1)}.")

### Not Applicable

#When day of week when fault occurs is selected as ‘not faulted’,

runner.registerAsNotApplicable('Measure NoOvernightSetbackWeek not run')

#When lighting schedule is not properly defined in the model,  
runner.registerAsNotApplicable("No changes made, selected zones may not have had setpoint schedules, or they schedules may not have been ScheduleRulesets.")

### Warning

n/a

### Error

n/a

### Information

Following measures share the same (or similar) functions.

* LightingSetbackErrorDelayedOnset
* LightingSetbackErrorEarlyTermination
* LightingSetbackErrorNoOvernightSetback

### 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 lighting schedules.
* Gather lighting schedule fraction values from those schedules (minimum and maximum).
* Create faulted schedule based on input arguments reflecting no overnight 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
* Assign modified (or faulted) lighting schedule(s) to assigned zone(s).

### 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.