## Improper Time Delay Setting in Occupancy Sensors

### Description

def description

return “Compared to scheduled lighting operation, using an occupancy sensor for the lighting control is more suitable when the space is intermittently occupied. In other words, when the space is left with the lights on for large amount of portion throughout the day, it is better to use the occupancy sensor to save the lighting energy consumption. However, setting a time delay in the occupancy sensor is a trade-off between occupant’s visual discomfort and energy savings. If the time delay is too short, chances increase for energy savings. But on the other side, lights being on and off too often increases visual discomfort for occupants in the space. 15 minutes of time delay is common in the real application, however, the setting can be improperly implemented in the field. This fault measure is based on a physical model where certain parameter(s) is changed in EnergyPlus to mimic the faulted operation; thus simulates the effect of an improper time delay setting in occupancy sensors by modifying the Schedule:Compact object in EnergyPlus assigned to lighting schedules. This fault is categorized as a fault that occur in the lighting system (controller) during the operation stage. The fault intensity (F) is defined as the delayed time setting (in hours).”

end

### Modeler Description

def modeler\_description

return “The measure detects the original occupancy schedule applied in EnergyPlus, and adjusts the lighting schedule assigned to the selected zone according to the occupancy schedule with the time delay applied based on the user inputs. Five different user inputs are required to simulate the fault; 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, time delay in hours.”

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 delayed.'

)

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 occupancy schedule affected by the fault.

runner.registerInitialCondition("Initial occupancy 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 affected by the fault.

runner.registerFinalCondition("Final occupancy 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 schedule is not applied properly,

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

#When fault intensity is defined as zero,  
runner.registerAsNotApplicable('Zero hour extension in Measure ' \

'Improper Time Delay Setting in Occupancy Sensors. ' \

'Exiting......')

### Warning

n/a

### Error

n/a

### Information

* Reads occupancy schedule and modify the schedule based on fault intensity and apply modified schedule to lighting schedule. Based on the assumption that lighting control based on occupancy sensor is simulated by using the occupancy schedule in lighting schedule field.
* Future refinement item is,
  + Capability to work with multimodal occupancy profiles.

### 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\_occupancy
    - Create default schedule... createnewdefaultdayofweekrule\_occupancy
      * 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\_occupancy
        + Find building opening time or closing time... findchangetime
        + Create schedule according to faulted time period... newtimesandvaluestosceduleday\_occupancy

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\_occupancy
        + 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.