## Return Air Duct Leakages

### Description

def description

return "The return duct of an air system typically operates at negative pressure, thus the leakage in the return duct (outside of conditioned space) results in increased heating and cooling load due to unconditioned air being drawn into the return duct and mixing with return air from conditioned spaces. This measure simulates the return air leakage by modifying the Controller:OutdoorAir object in EnergyPlus. The fault intensity (F) for this fault is defined as the unconditioned air introduced to return air stream at full load condition as a ratio of the total return airflow rate."

end

### Modeler Description

def modeler\_description

return "Nine user inputs are required to simulate the fault and, based on these inputs, this fault model simulates the return air duct leakage by introducing additional outdoor air (based on the leakage ratio) through the economizer object. Equation (2) shows the calculation of outdoor airflow rate in the economizer (qdot\_(oa,F)) at a faulted condition where qdot\_oa is the outdoor airflow rate for ventilation, qdot\_(ra,tot) is the return airflow rate, F is the fault intensity and AF is the adjustment factor. qdot\_(oa,F) = qdot\_oa + qdot\_(ra,tot)∙F∙AF. The second term represents the outdoor airflow rate introduced to the duct due to leakage. The fault intensity (F) for this fault is defined as the unconditioned air introduced to return air stream at full load condition as a ratio of the total return airflow rate. 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

EnergyPlus Measure

**Taxonomy**

HVAC.Ventilation

### Arguments

def arguments(workspace)

args = OpenStudio::Ruleset::OSArgumentVector.new

#make choice arguments for economizers

controlleroutdoorairs = workspace.getObjectsByType("Controller:OutdoorAir".to\_IddObjectType)

chs = OpenStudio::StringVector.new

controlleroutdoorairs.each do |controlleroutdoorair|

chs << controlleroutdoorair.name.to\_s

end

econ\_choice = OpenStudio::Ruleset::OSArgument::makeChoiceArgument('econ\_choice', chs, true)

econ\_choice.setDisplayName('Outdoor air controller affected by the leakage of unconditioned air from the ambient.')

econ\_choice.setDefaultValue(chs[0].to\_s)

args << econ\_choice

#make a double argument for the return duct leakage

leak\_ratio = OpenStudio::Ruleset::OSArgument::makeDoubleArgument('leak\_ratio', false)

leak\_ratio.setDisplayName('Enter the unconditioned air introduced to return air stream at full load condition as a ratio of the total return airflow rate [0-1].')

leak\_ratio.setDefaultValue(0.1) #default fault level to be 10%

args << leak\_ratio

#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::makeDoubleArgument('start\_month', false)

start\_month.setDisplayName('Enter the month (1-12) when the fault starts to occur')

start\_month.setDefaultValue(6) #default is June

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 start time

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

start\_time.setDisplayName('Enter the time of day (0-24) when the fault starts to occur')

start\_time.setDefaultValue(9) #default is 9am

args << start\_time

#make a double argument for the end month

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

end\_month.setDisplayName('Enter the month (1-12) when the fault ends')

end\_month.setDefaultValue(12) #default is Decebmer

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

#make a double argument for the end time

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

end\_time.setDisplayName('Enter the time of day (0-24) when the fault ends')

end\_time.setDefaultValue(23) #default is 11pm

args << end\_time

return args

end

### Initial Condition

#Select economizer object that is being faulted.

runner.registerInitialCondition("Imposing Return Duct Leakage on #{econ\_choice}.")

### Final Condition

#Duct leakage applied to the economizer object.

runner.registerFinalCondition("Imposed Return Duct Leakage on #{econ\_choice}.")

### Not Applicable

#When fault intensity is defined as zero,

runner.registerAsNotApplicable("#{name} is not running with zero fault level. Skipping......")

#MinimumFlowWithBypass option in Controller:OutdoorAir is not supported.

runner.registerAsNotApplicable("MinimumFlowWithBypass in #{econ\_choice} is not supported. Skipping......")

#LockoutWithHeating & LockoutWithCompressor options in Controller:OutdoorAir are not supported.

runner.registerAsNotApplicable(controlleroutdoorair.getString(14).to\_s+" in #{econ\_choice} is not supported. Skipping......")

#BypassWhenOAFlowGreaterThanMinimum option in Controller:OutdoorAir is not supported.

runner.registerAsNotApplicable(controlleroutdoorair.getString(25).to\_s+" in #{econ\_choice} is not supported. Skipping......")

#When Controller:OutdoorAir was not selected properly,

runner.registerAsNotApplicable("#{name} is not running for #{econ\_choice} because of inapplicability. Skipping......")

### Warning

n/a

### Error

#user selected Controller:OutdoorAir cannot be found,

runner.registerError("Measure #{name} cannot find #{econ\_choice}. Exiting......")

### Information

* Calculates required OA flow rate at given timestep. Heavy code. Works with Controller:OutdoorAir.
* The code is similar to Biased Economizer Sensor Faults.
* Future refinement item is,
  + Capability to work with other biased economizer sensor offset faults.

### Code Outline

* Define arguments (economizer where fault occurs, fault level in constant value etc).
* Find the economizer where the fault occurs.
* Check applicability of the model to the economizer defined in the model.
* Write EMS program (appropriately according to economizer options that were already defined in the model) to impose return duct leakage for each economizer object.
* Append EMS code to impose leakage ratio level at the outdoor air measurement reading.
* Append appropriate EMS code if Minimum Outdoor Air Schedule option is defined.
* Append appropriate EMS code if Mechanical Ventilation Controller option is defined.
* Append appropriate EMS code if Economizer Control Type option is defined as NoEconomizer. If not,
  + Append appropriate EMS code if Lockout Type option is defined.
    - Append appropriate EMS code if Lockout Type option is defined as either LockoutWithHeating or LockoutWithCompressor.
  + Append appropriate EMS code if Economizer Control Type option is defined as DifferentialDryBulb.
  + Append appropriate EMS code if Economizer Control Type option is defined as either FixedDryBulb, FixedEnthalpy, FixedDewPointAndDryBulb or ElectronicEnthalpy.
  + Append appropriate EMS code if Economizer Control Type option is defined as DifferentialDryBulbAndEnthalpy.
  + Append appropriate EMS code if Economizer Control Type option is defined as DifferentialEnthalpy.
  + Append appropriate EMS code if Economizer Minimum Limit Dry-Bulb Temperature option is defined.
  + Append appropriate EMS code if High Humidity Control option is defined as yes.
    - Append appropriate EMS code if Control High Indoor Humidity Based on Outdoor Humidity Ratio option is defined as yes.
  + Append appropriate EMS code if Time of Day Economizer Control Schedule Name option is defined.
* Append appropriate EMS code if Economizer Control Action Type option is defined as MinimumFlowWithBypass.
* Append appropriate EMS code if High Humidity Control option is defined as yes.
* Append appropriate EMS code if Minimum Fraction of Outdoor Air Schedule Name option is defined.
* Append appropriate EMS code if Maximum Fraction of Outdoor Air Schedule Name option is defined.
* Append appropriate EMS code to calculate modified outdoor air flow rate.
* Append appropriate EMS code to check whether modified outdoor air flow rate exceeds maximum limit.
* Append appropriate EMS code and texts for defining objects in idf based on above options to complete the code.
  + Define EnergyManagementSystem:Subroutine
  + Define EnergyManagementSystem:ProgramCallingManager
  + Define EnergyManagementSystem:GlobalVariable
  + Define EnergyManagementSystem:Actuator
  + Define EnergyManagementSystem:InternalVariable
  + Define EnergyManagementSystem:Sensor
  + Define Output:EnergyManagementSystem
* Append EMS code that calculates the adjustment factor (AF)

### Tests

* Test model with different economizer object types
* Test invalid user argument values to make sure measure fails gracefully