## 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."

end

### Modeler Description

def workspaceer\_description

return "Two user inputs (economizer included in the air terminal unit where the fault occurs, unconditioned air introduced to return air stream at full load condition as a ratio of the total airflow rate, F) 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, and F is the fault intensity. qdot\_(oa,F) = qdot\_oa + qdot\_(ra,tot)∙F. 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."

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("Choice of economizers.")

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

args << econ\_choice

# make a double argument for the leakage ratio

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

leak\_ratio.setDisplayName('Ratio of leak airflow between 0 and 0.3.')

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

args << leak\_ratio

return args

end

### Initial Condition

#Select economizer object that is being faulted.

runner.registerInitialCondition("Imposing duct leakages on #{econ\_choice}.")

### Final Condition

#Duct leakage applied to the economizer object.

runner.registerFinalCondition("Imposed performance degradation on #{econ\_choice}.")

### Not Applicable

#When fault intensity is defined as zero,

runner.registerAsNotApplicable("ReturnAirDuctLeakages is not running for "+econ\_choice+". Skipping......")

### Warning

n/a

### Error

#When fault intensity is defined outside of the range 0-30%,

runner.registerError("Fault level #{leak\_ratio} for #{econ\_choice} is outside the range from 0 to 0.3. Exiting......")

#When selected economizer cannot be found in the model,

runner.registerError("Measure ReturnAirDuctLeakages cannot find "+econ\_choice+". Exiting......")

### Information

n/a

### Code Outline

* Define arguments (economizer where fault occurs, fault level in constant value).
* Check constant fault level value (within 0-0.3).
* Find economizer and add EMS code to override the outdoor air flow rate
  + Add EnergyManagementSystem:Sensor
    - Read total return air flow rate of the air system
  + Add EnergyManagementSystem:InternalVariable
    - Read minimum outdoor air flow rate of the economizer
  + Add EnergyManagementSystem:Program
    - Caculate increased outdoor air flow rate based on the fault intensity, qdot\_(oa,F) = qdot\_oa + qdot\_(ra,tot)∙F
  + Add EnergyManagementSystem:ProgramCallingManager
  + Add EnergyManagementSystem:Actuator
    - Impose modified outdoor air flow rate to the economizer

### Tests

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