1. Window (IOT 24)

|  |
| --- |
| from gpio import \*  from time import \*  from ioeclient import \*  from physical import \*  from environment import \*  import math  ENVIRONMENTS = ["Argon", "CO", "CO2", "Hydrogen", "Helium", "Methane", "Nitrogen", "O2", "Ozone", "Propane", "Smoke"]  ENVIRONMENT\_MAX\_IMPACT = -0.01; # 2% max when door opens  TEMPERATURE\_TRANSFERENCE\_MULTIPLIER = 1.20; # increase speed 25% when door open  HUMIDITY\_TRANSFERENCE\_MULTIPLIER = 1.20  GASES\_TRANSFERENCE\_MULTIPLIER = 2  state = 0  def main():  setup()  while True:  loop()  #set up client to talk and listen to IoE registration server  def setup():  IoEClient.setup({  "type": "Window",  "states": [{  "name": "On",  "type": "bool",  "controllable": True  }]  })  IoEClient.onInputReceive(onInputReceiveDone)  add\_event\_detect(0, detect)    state = restoreProperty("state", 0)  setState(state)  def onInputReceiveDone(data):  processData(data, True)    def detect():  processData(customRead(0), false)    def restoreProperty(propertyName, defaultValue):  value = getDeviceProperty(getName(), propertyName)  if not (value is "" or value is None):  if type(defaultValue) is int :  value = int(value)  setDeviceProperty(getName(), propertyName, value)  return value  return defaultValue  def mouseEvent(pressed, x, y, firstPress):  global state  if firstPress:  if state == True:  setState(0)  else:  setState(1)  #update carbon dioxide and carbon monoxide and send new data to registration server  def loop():  updateEnvironment()  delay(1000)  #process data received from server  def processData(data, bIsRemote):  if len(data) <= 0 :  return  data = data.split(",")  setState(int(data[0]))  #set state and update component image to reflect the current state  def setState(newState):  global state  if newState is 0 :  digitalWrite(1, LOW)  else:  digitalWrite(1, HIGH)    state = newState  customWrite(0, state)  IoEClient.reportStates(state)  setDeviceProperty(getName(), "state", state)  def updateEnvironment():  global ENVIRONMENTS  global ENVIRONMENT\_MAX\_IMPACT  global GASES\_TRANSFERENCE\_MULTIPLIER  global TEMPERATURE\_TRANSFERENCE\_MULTIPLIER  global HUMIDITY\_TRANSFERENCE\_MULTIPLIER  global state  if state == 1:  for i in range (0,len(ENVIRONMENTS)):  max = Environment.get(ENVIRONMENTS[i]) \* ENVIRONMENT\_MAX\_IMPACT  # the max is reached in an hour, so we divide by 3600 to get seconds  # then this rate is also based on 100,000 cubic meters (approx. coporate office size)  rate = float(max) / 3600 \* 100000 / Environment.getVolume()  Environment.setContribution(ENVIRONMENTS[i], rate, max, True)  Environment.setTransferenceMultiplier(ENVIRONMENTS[i], GASES\_TRANSFERENCE\_MULTIPLIER)  Environment.setTransferenceMultiplier("Ambient Temperature", TEMPERATURE\_TRANSFERENCE\_MULTIPLIER)  Environment.setTransferenceMultiplier("Humidity", HUMIDITY\_TRANSFERENCE\_MULTIPLIER)  else:  for j in range (0, len(ENVIRONMENTS)):  Environment.setContribution(ENVIRONMENTS[j], 0, 0, True)  Environment.removeCumulativeContribution(ENVIRONMENTS[j])  Environment.setTransferenceMultiplier(ENVIRONMENTS[j], 1)  Environment.setTransferenceMultiplier("Ambient Temperature", 1)  Environment.setTransferenceMultiplier("Humidity", 1)  if \_\_name\_\_ == "\_\_main\_\_":  main() |

2. Air-Conditioner (IOT 36)

|  |
| --- |
| from gpio import \*  from time import \*  from ioeclient import \*  from environment import \*  from physical import \*  HUMIDITY\_RATE = -2./3600; # -2% per hour  TEMPERATURE\_RATE = -10./3600; # -10C per hour  VOLUME\_AT\_RATE = 100000.  myInput = 0    def onInputReceiveDone(data):  if len(data) <= 0:  return  data = data.split(",")  processData(int(data[0]))    def detect():  processData(digitalRead(0)/1023)    def setup():  IoEClient.setup({  "type": "AC",  "states": [{  "name": "On",  "type": "bool",  "controllable": True  }]  })  IoEClient.onInputReceive(onInputReceiveDone)  add\_event\_detect(0, detect)  detect()    VAR = getDeviceProperty(getName(), "VOLUME\_AT\_RATE")  if not VAR:  setDeviceProperty(getName(), "VOLUME\_AT\_RATE", VOLUME\_AT\_RATE)  def processData(data):  global myInput  myInput = data  if myInput > 0 :  digitalWrite(5, HIGH)  else :  digitalWrite(5, LOW)  IoEClient.reportStates(myInput)  def updateEnvironment():  VAR = float(getDeviceProperty(getName(), "VOLUME\_AT\_RATE"))  if VAR < 0:  VAR = 0  humidity\_rate = float(myInput\*HUMIDITY\_RATE\*VAR) / Environment.getVolume()  temperature\_rate = float(myInput\*TEMPERATURE\_RATE\*VAR) / Environment.getVolume()  Environment.setContribution("Humidity", humidity\_rate, 0, True)  Environment.setContribution("Ambient Temperature", temperature\_rate, -1000, True)  #print temperature\_rate, VAR  def main():  setup()  while True:  updateEnvironment()  delay(1000)    if \_\_name\_\_ == "\_\_main\_\_":  main() |

3. Ceiling-Fan (IOT 0): -

|  |
| --- |
| from gpio import \*  from time import \*  from ioeclient import \*  from physical import \*  from environment import \*  import math  FAN\_SPEED\_LOW = 0.4; # kph  FAN\_SPEED\_HIGH = 0.8; # kph  COOLING\_RATE = float(-1)/3600; # -1C/hour  HUMDITY\_REDUCTION\_RATE = float(-1)/3600; # -1%/hour  VOLUME\_AT\_RATE = 100000; # the given rates are based on this volume  state = 0; # 0 off, 1 low, 2 high  level = 0  def main():  global state  IoEClient.setup({  "type": "Ceiling Fan",  "states": [  {  "name": "Status",  "type": "options",  "options": {  "0": "Off",  "1": "Low",  "2": "High"  },  "controllable": True  }  ]  })  IoEClient.onInputReceive(onInputReceiveDone)  add\_event\_detect(0, detect)  state = restoreProperty("state", 0)  setState(state)  while True:  delay(1000)  def onInputReceiveDone(data):  processData(data, True)    def detect():  processData(customRead(0), False)    def restoreProperty(propertyName, defaultValue):  value = getDeviceProperty(getName(), propertyName)  if not (value is "" or value is None):  if type(defaultValue) is int :  value = int(value)  setDeviceProperty(getName(), propertyName, value)  return value  return defaultValue  def mouseEvent(pressed, x, y, firstPress):  if firstPress:  toggleState()  def processData(data, bIsRemote):  if len(data) <= 0 :  return  data = data.split(",")  setState(int(data[0]))  def sendReport():  global state  global report  report = state # comma seperated states  customWrite(0, report)  IoEClient.reportStates(report)  setDeviceProperty(getName(), "state", state)  def setState(newState):  global state  analogWrite(A1, newState)  state = newState  sendReport()  updateEnvironment()  def toggleState():  global state  state += 1  if int(state) >= 3:  state = 0  setState(state)  def updateEnvironment():  global VOLUME\_AT\_RATE  global FAN\_SPEED\_LOW  global COOLING\_RATE  global HUMDITY\_REDUCTION\_RATE  global FAN\_SPEED\_HIGH  global state  volumeRatio = float(VOLUME\_AT\_RATE) / Environment.getVolume()  if int(state) == 0:  Environment.setContribution("Wind Speed", 0, 0, True)  Environment.setContribution("Ambient Temperature", 0, 0, True)  Environment.setContribution("Humidity", 0,0, True)  elif int(state) == 1:  Environment.setContribution("Wind Speed", FAN\_SPEED\_LOW, FAN\_SPEED\_LOW, False)  # everytime the fan restarts, it can do another -100C  Environment.setContribution("Ambient Temperature", float(COOLING\_RATE)/2\*volumeRatio, Environment.getCumulativeContribution("Ambient Temperature")-100, True)  Environment.setContribution("Humidity", float(HUMDITY\_REDUCTION\_RATE)/2\*volumeRatio, Environment.getCumulativeContribution("Humidity")-100, True)  elif int(state) == 2:  Environment.setContribution("Wind Speed", FAN\_SPEED\_HIGH, FAN\_SPEED\_HIGH, False)  Environment.setContribution("Ambient Temperature", float(COOLING\_RATE)/2\*volumeRatio, Environment.getCumulativeContribution("Ambient Temperature")-100, True)  Environment.setContribution("Humidity", HUMDITY\_REDUCTION\_RATE\*volumeRatio, Environment.getCumulativeContribution("Humidity")-100, True)  if \_\_name\_\_ == "\_\_main\_\_":  main() |

4. Beacon (IOT 22):-

|  |
| --- |
| from physical import \*  from bluetooth import \*  from time import \*  DEFAULT\_BEACON\_UUID = "{00000000-0000-0000-0000-000000000001}"  DEFAULT\_BEACON\_DATA = "Location 1"  def setup():  Bluetooth.init()  Bluetooth.enableBroadcast(True)  global DEFAULT\_BEACON\_UUID  global DEFAULT\_BEACON\_DATA  uuid = getDeviceProperty(getName(), "beaconUuid")  if uuid == None:  setDeviceProperty(getName(), "beaconUuid", DEFAULT\_BEACON\_UUID)  data = getDeviceProperty(getName(), "beaconData")  if data == None:  setDeviceProperty(getName(), "beaconData", DEFAULT\_BEACON\_DATA)  def main():  setup()  while True:  uuid = getDeviceProperty(getName(), "beaconUuid")  data = getDeviceProperty(getName(), "beaconData")  print "Broadcasting to " + str(uuid)  Bluetooth.broadcastBeacon(uuid, data)  delay(5000)    if \_\_name\_\_ == "\_\_main\_\_":  main() |

5. Light (IOT 10):-

|  |
| --- |
| from gpio import \*  from time import \*  from physical import \*  from ioeclient import \*  from environment import \*  ENVIRONMENT\_IMPACT\_DIM = 10  VOLUME\_AT\_RATE = 100000  state = 0; # 0 off, 1 low, 2 high  lastTimeInSeconds = 0  def main():  setup()  while True:  loop()    def setup():  IoEClient.setup({  "type": "Light",  "states": [  {  "name": "Status",  "type": "options",  "options": {  "0": "Off",  "1": "Dim",  "2": "On"  },  "controllable": True  }  ]  })  IoEClient.onInputReceive(onInputReceiveDone)  global state  add\_event\_detect(0, detect)  state = restoreProperty("state", 0)  setState(state)  def detect():  processData(customRead(0), False)    def onInputReceiveDone(analogInput):  processData(analogInput, True)    def restoreProperty(propertyName, defaultValue):  value = getDeviceProperty(getName(), propertyName)  if not (value is "" or value is None):  if type(defaultValue) is int :  value = int(value)  setDeviceProperty(getName(), propertyName, value)  return value  return defaultValue  def mouseEvent(pressed, x, y, firstPress):  global state  if firstPress:  setState(state+1)  def loop():  updateEnvironment()  sleep(1)  def processData(data, bIsRemote):  if len(data) <= 0 :  return  setState(int(data))  def setState(newState):  global state  if newState >= 3 :  newState = 0  state = newState  analogWrite(A1, state)  customWrite(0, state)  IoEClient.reportStates(state)  setDeviceProperty(getName(), "state", state)  def updateEnvironment():  global VOLUME\_AT\_RATE  global ENVIRONMENT\_IMPACT\_DIM  volumeRatio = float(VOLUME\_AT\_RATE) / Environment.getVolume()  if state is 0 :  Environment.setContribution("Visible Light", 0,0, True)  elif state is 1:  Environment.setContribution("Visible Light", ENVIRONMENT\_IMPACT\_DIM\*volumeRatio, ENVIRONMENT\_IMPACT\_DIM\*volumeRatio, False)  elif state is 2 :  Environment.setContribution("Visible Light", ENVIRONMENT\_IMPACT\_DIM\*2\*volumeRatio, ENVIRONMENT\_IMPACT\_DIM\*2\*volumeRatio, False)  if \_\_name\_\_ == "\_\_main\_\_":  main() |

6. Solar-Panel (IOT 32): -

|  |
| --- |
| from options import Options  from time import \*  import math  from physical import \*  from gpio import \*  from environment import Environment  from ioeclient import IoEClient  #from pyjs import \*  #Solar Panel  #Read the sunlight levels  #Output electricity based on sunlight  #Panel will be 160Watts per square meter  #Features output to IoE Server:  # number of kWh of energy produced since turning on  # number of kWh per minute  # current production  ENVIRONMENT\_NAME = "Sunlight" # var ENVIRONMENT\_NAME  MULTIPLIER = 255. / 1023 # var MULTIPLIER  MAX\_POWER = 1000. #1000 Watts of power based on one meter solar panel at noon at the equator # var MAX\_POWER  EFFICIENCY = 0.16 #About a 16 percent efficiency per solar panel # var EFFICIENCY  PANEL\_POWER = MAX\_POWER \* EFFICIENCY # var PANEL\_POWER  LOG\_BASE = 1.0749111034571373359815489867558 # var LOG\_BASE  state = 1 # var state  electricity = 0 # var electricity  #tick = 0 # var tick  def setup ():    IoEClient.setup({  "type": "Solar",  "states": [{  "name": "Status",  "type": "number",  "unit": 'Wh',  "controllable": False  }]  })  IoEClient.onInputReceive ( lambda rinput: processData(rinput, True) )  sendReport()  def loop ():  global electricity  ## if (tick++ % 10) is 0 ) # is tick consistent across devices?  ## {  electricity = int(getElectricityProduction())  ##print(electricity)  displayElectricity()  sendReport()  outputElectricity()  delay(1000)  ##  def displayElectricity ():  setCustomText(70, 45, 1000, 1000, str(int(electricity)) + '\tW')  def getElectricityProduction ():  return PANEL\_POWER \* Environment.get(ENVIRONMENT\_NAME) / 100  def sendReport ():  report = state # comma seperated states # var report  IoEClient.reportStates(electricity)  setDeviceProperty(getName(), "level", electricity)  def outputElectricity ():  el\_log = math.floor(math.log(electricity) / math.log(LOG\_BASE)) # var el\_log  if el\_log < 0:  el\_log = 0  elif el\_log > 255:  el\_log = 255  ## print(el\_log)  analogWrite(0, el\_log)  if \_\_name\_\_ == "\_\_main\_\_":  setup()  while True:  loop()  sleep(0) |

7. Wind-Detector (IOT 33): -

|  |
| --- |
| from options import Options  from time import \*  import math  from physical import \*  from gpio import \*  from environment import Environment  from ioeclient import IoEClient  ENVIRONMENT\_NAME = "Wind Speed" # var ENVIRONMENT\_NAME  state = 0 # var state  tick = 0 # var tick  # set up client to talk and listen to IoE registration server  def setup ():    IoEClient.setup({  "type": "Wind Detector",  "states": [{  "name": "Wind",  "type": "bool",  "controllable": False  }]  })    IoEClient.onInputReceive ( lambda rinput: processData(input, True) )  setState(state)  sendReport()  # continously checking if WIND exist and send report to registration server  def loop ():  global tick  if tick % 10 == 0: # is tick consistent across devices?  detect()  sendReport()  tick += 1    # get WIND variable defined in Environment  def detect ():  value = Environment.get(ENVIRONMENT\_NAME) # var value  if value >= 1:  setState(1)  else:  setState(0)  # process data received from server  # not being called since controllable set to False in client setup  def processData (data, bIsRemote):  if len(data) <= 0:  return  data = data.split(",") # var data  setState(int(data[0]))  # send wind state to the server  def sendReport ():  report = state # comma seperated states # var report  IoEClient.reportStates(report)  # set state and update component image to reflect the current state  def setState (newState):  global state  if newState == 0:  digitalWrite(1, LOW)  else:  digitalWrite(1, HIGH)  state = newState  sendReport()  # toggle wind state  def toggleState ():  if state == 0:  setState(1)  else:  setState(0)  if \_\_name\_\_ == "\_\_main\_\_":  setup()  while True:  loop()  sleep(0) |

8. Door : -

|  |
| --- |
| from time import \*  from physical import \*  from gpio import \*  from environment import Environment  from ioeclient import IoEClient  ENVIRONMENTS = ["Argon", "CO", "CO2", "Hydrogen", "Helium", "Methane", "Nitrogen", "O2", "Ozone", "Propane", "Smoke"]  ENVIRONMENT\_MAX\_IMPACT = -0.02 # 2% max when door opens  TEMPERATURE\_TRANSFERENCE\_MULTIPLIER = 1.25 # increase speed 25% when door open  HUMIDITY\_TRANSFERENCE\_MULTIPLIER = 1.25  GASES\_TRANSFERENCE\_MULTIPLIER = 2  doorState = 0 # 0 is closed, 1 is opened  lockState = 0 # 0 is unlocked, 1 is locked  def on\_event\_detect\_0 () :  processData(customRead(0), False)  def on\_input\_receive(input) :  processData(input, True)  def setup ():  IoEClient.setup({  "type": "Door",  "states": [{  "name": "Open",  "type": "bool"  },  {  "name": "Lock",  "type": "options",  "options": {  0: "Unlock",  1: "Lock"  },  "controllable": True  }]  })  IoEClient.onInputReceive(on\_input\_receive)  add\_event\_detect(0, on\_event\_detect\_0)  setDoorState(doorState)  setLockState(lockState)  def mouseEvent (pressed, x, y, firstPress):  if firstPress:  if isPointInRectangle(x, y, 10,40,5,10) :  if lockState == 0 :  setLockState(1)  else:  setLockState(0)  else:  if doorState == 0 :  openDoor()  else:  closeDoor()  def processData (data, bIsRemote):  if len(data) < 1 :  return  print data  data = data.split(",")  doorStateData = int(data[0])  lockStateData = int(data[1])  if lockStateData > -1 :  setLockState(lockStateData)  if doorStateData > -1 and not bIsRemote :  if doorStateData == 0 :  closeDoor()  else:  openDoor()  def sendReport ():  report = str(doorState)+","+str(lockState) # comma seperated states  customWrite(0, report)  IoEClient.reportStates(report)  setDeviceProperty(getName(), "door state", doorState)  setDeviceProperty(getName(), "lock state", lockState)  def closeDoor ():  setDoorState(0)  updateEnvironment()  def openDoor ():  if lockState == 0 :  setDoorState(1)  updateEnvironment()  else:  print "can't open locked door"  def setDoorState (state):  global doorState  if state == 0:  digitalWrite(1, LOW)  setComponentOpacity("led", 1) # show the led  else:  digitalWrite(1, HIGH)  setComponentOpacity("led", 0) # hide the led  doorState = state  sendReport()  def setLockState (state):  global lockState  if state == 0 :  digitalWrite(2, LOW)  else:  digitalWrite(2, HIGH)  lockState = state  sendReport()  def updateEnvironment ():  rate, emax = 0, 0  if doorState == 1:  for e in ENVIRONMENTS:  emax = Environment.get(e) \* ENVIRONMENT\_MAX\_IMPACT  # the emax is reached in an hour, so we divide by 3600 to get seconds  # then this rate is also based on 100,000 cubic meters (approx. coporate office size)  rate = emax / 3600 \* 100000 / Environment.getVolume()  Environment.setContribution(e, rate, emax, True)  Environment.setTransferenceMultiplier(e, GASES\_TRANSFERENCE\_MULTIPLIER)  Environment.setTransferenceMultiplier("Ambient Temperature", TEMPERATURE\_TRANSFERENCE\_MULTIPLIER)  Environment.setTransferenceMultiplier("Humidity", HUMIDITY\_TRANSFERENCE\_MULTIPLIER)  else:  for e in ENVIRONMENTS:  Environment.setContribution(e, 0, 0, True)  Environment.removeCumulativeContribution(e)  Environment.setTransferenceMultiplier(e, 1)  Environment.setTransferenceMultiplier("Ambient Temperature", 1)  Environment.setTransferenceMultiplier("Humidity", 1)  def isPointInRectangle (x,y, rx, ry, width, height):  if width <= 0 or height <= 0:  return False  return (x >= rx and x <= rx + width and y >= ry and y <= ry + height)  if \_\_name\_\_ == "\_\_main\_\_":  setup()  while True:  sleep(0) |

9. Appliance: -

|  |
| --- |
| from gpio import \*  from time import \*  from ioeclient import \*  from environment import \*  from physical import \*  state = 0  def main():  setup()  global state  while True:  state = restoreProperty("state", 0)  setState(state)  delay(1000)    def setup():  IoEClient.setup({  "type": "Appliance",  "states": [{  "name": "On",  "type": "bool",  "controllable": True  }]  })  IoEClient.onInputReceive(onInputReceiveDone)  add\_event\_detect(0, detect)    def onInputReceiveDone(data):  processData(data, True)    def detect():  processData(customRead(0), False)    def restoreProperty(propertyName, defaultValue):  value = getDeviceProperty(getName(), propertyName)  if not (value is "" or value is None):  if type(defaultValue) is int :  value = int(value)  setDeviceProperty(getName(), propertyName, value)  return value  return defaultValue  def mouseEvent(pressed, x, y, firstPress):  global state  if firstPress:  if state == True:  setState(0)  else:  setState(1)  def processData(data, bIsRemote):  if len(data) <= 0 :  return  setState(int(data))  def setState(newState):  global state  state = newState  if state == 0 :  digitalWrite(1, LOW)  else:  digitalWrite(1, HIGH)  customWrite(0, state)  IoEClient.reportStates(state)  setDeviceProperty(getName(), "state", state)  if \_\_name\_\_ == "\_\_main\_\_":  main() |

10. Water Drain: -

|  |
| --- |
| from time import \*  import math  from physical import \*  from gpio import \*  from environment import Environment  from ioeclient import IoEClient  WATERLEVEL\_RATE = -0.5 # 0.5 cm per second # var WATERLEVEL\_RATE  VOLUME\_AT\_RATE = 100000/2.5 # var VOLUME\_AT\_RATE  MAX\_RATE = -1.e6 # var MAX\_RATE  state = 0 # 0 off, 1 on # var state  def setup ():    IoEClient.setup({  "type": "Water Drain",  "states": [{  "name": "Status",  "type": "bool",  "controllable": True  }]  })  IoEClient.onInputReceive = lambda rinput: processData(rinput, True)  def on\_event\_detect():  processData(customRead(0), False)  add\_event\_detect(0, on\_event\_detect)  setState(state)  def mouseEvent (pressed, x, y, firstPress):  if firstPress:  setState(( 0 if state else 1 ) )  def processData (data, bIsRemote):  if len(data) <= 0:  return  setState(int(data))  def setState (newState):  global state  state = newState  analogWrite(A1, state)  customWrite(0, state)  IoEClient.reportStates(state)  setDeviceProperty(getName(), "state", state)  updateEnvironment()  def updateEnvironment ():  if state == 1:  volumeRatio = VOLUME\_AT\_RATE / Environment.getVolume() # var volumeRatio  if Environment.get("Water Level") >= 0:  Environment.setContribution("Water Level", WATERLEVEL\_RATE \* volumeRatio, MAX\_RATE, True)  else:  Environment.setContribution("Water Level", 0, 0, True)  else:  Environment.setContribution("Water Level", 0, 0, True)    if \_\_name\_\_ == "\_\_main\_\_":  setup()  while True:  #loop()  sleep(0) |

11. Smoke-Detector: -

|  |
| --- |
| from gpio import \*  from time import \*  from ioeclient import \*  from physical import \*  import math  from environment import \*  ENVIRONMENT\_NAME = "Smoke"  state = 0  level = 0  ALARM\_LEVEL = 40  def main():  setup()  while True:  loop()    def setup():  IoEClient.setup({  "type": "Smoke Detector",  "states": [{  "name": "Alarm",  "type": "bool",  "controllable": False  },  {  "name": "Level",  "type": "number",  "controllable": False  }]  })  restoreProperty("Alarm Level", 40)  IoEClient.onInputReceive(onInputReceiveDone)  add\_event\_detect(0, detect)    state = restoreProperty("state", 0)  setState(state)  def onInputReceiveDone(data):  processData(data, True)    def detect():  processData(customRead(0), False)  def restoreProperty(propertyName, defaultValue):  value = getDeviceProperty(getName(), propertyName)  if not (value is "" or value is None):  if type(defaultValue) is int :  value = int(value)  setDeviceProperty(getName(), propertyName, value)  return value  return defaultValue  def loop():  global ENVIRONMENT\_NAME  value = Environment.get(ENVIRONMENT\_NAME)  if value >= 0:  setLevel(Environment.get(ENVIRONMENT\_NAME))  #print(value)  sleep(1)  def processData(data, bIsRemote):  if len(data) <= 0 :  return  data = data.split(",")  setState(int(data[0]))  def sendReport():  global state  global level  report = str(state) + "," + str(level); # comma seperated states  IoEClient.reportStates(report)  setDeviceProperty(getName(), "state", state)  setDeviceProperty(getName(), "level", level)  def setState(newState):  global state  state = newState  if newState is 0:  digitalWrite(1, LOW)  else:  digitalWrite(1, HIGH)  sendReport()  def setLevel(newLevel):  global level  if level == newLevel:  return  level = newLevel  if level > ALARM\_LEVEL:  setState(1)  else:  setState(0)  sendReport()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

12. Web- Cam: -

|  |
| --- |
| from gpio import \*  from time import \*  from ioeclient import \*  from physical import \*  from environment import \*  import math  state = 0  def main():  setup()  while True:  loop()    #set up client to talk and listen to IoE registration server  def setup():  global state  IoEClient.setup({  "type": "Webcam",  "states": [{  "name": "On",  "type": "bool",  "controllable": True  },  {  "name": "Image",  "type": "image"  }]  })  IoEClient.onInputReceive(onInputReceiveDone)  add\_event\_detect(0, detect)  state = restoreProperty("state", 0)  sendReport()  def onInputReceiveDone(data):  processData(data, True)    def detect():  processData(customRead(0), False)    def restoreProperty(propertyName, defaultValue):  value = getDeviceProperty(getName(), propertyName)  if not (value is "" or value is None):  if type(defaultValue) is int :  value = int(value)  setDeviceProperty(getName(), propertyName, value)  return value  return defaultValue  def mouseEvent(pressed, x, y, firstPress):  global state  if firstPress:  if state == True:  setState(0)  else:  setState(1)  #send captured image file path to registration server  def loop():  sendReport()  delay(1000)  #process data received from server  def processData(data, bIsRemote):  if len(data) <= 0 :  return  data = data.split(",")  setState(int(data[0]))  #send image path to server  imageLoop=0  def sendReport():  global state  global imageLoop  report = str(state) + "," # comma seperated states  if state is 0:  report += '../art/IoE/SmartDevices/camera\_off.png'  else:  report += '../art/IoE/SmartDevices/camera\_image'+ str(imageLoop)+'.png'  imageLoop = imageLoop + 1  if imageLoop >= 3:  imageLoop =0    customWrite(0, report)  IoEClient.reportStates(report)  setDeviceProperty(getName(), "state", state)  #set state and update component image to reflect the current state  def setState(newState):  global state  if newState is 0 :  digitalWrite(1, LOW)  else:  digitalWrite(1, HIGH)  state = newState    if \_\_name\_\_ == "\_\_main\_\_":  main() |

13. Lawn Sprinkler

|  |
| --- |
| from options import Options  from time import \*  import math  from physical import \*  from gpio import \*  from environment import Environment  from ioeclient import IoEClient  #from pyjs import \*  WATERLEVEL\_RATE = 0.1 # 0.1 cm per second # var WATERLEVEL\_RATE  HUMIDITY\_RATE = 5. / 3600 # 5% per hour # var HUMIDITY\_RATE  VOLUME\_AT\_RATE = 100000. # var VOLUME\_AT\_RATE  MAX\_RATE = 1.e6 # var MAX\_RATE  state = 0 # 0 off, 1 on # var state  def setup ():  global state    IoEClient.setup({  "type": "Lawn Sprinkler",  "states": [{  "name": "Status",  "type": "bool",  "controllable": True  }]  })    IoEClient.onInputReceive(lambda rinput: processData(rinput, True))  def on\_event\_detect():  processData(customRead(0), False)  add\_event\_detect(0, on\_event\_detect)  state = restoreProperty("state", 0)  setState(state)  def restoreProperty (propertyName, defaultValue):  value = getDeviceProperty(getName(), propertyName) # var value  if value:  if isinstance(defaultValue, (int, float)):  value = int(value)  setDeviceProperty(getName(), propertyName, value)  return value    return defaultValue  def mouseEvent (pressed, x, y, firstPress):  if firstPress:  setState(( 0 if state else 1 ) )  def processData (data, bIsRemote):  if len(data) <= 0:  return  setState(int(data))  def setState (newState):  global state  state = newState  digitalWrite(5, state)  customWrite(0, state)  IoEClient.reportStates(state)  setDeviceProperty(getName(), "state", state)  updateEnvironment()  def updateEnvironment ():  if state == 1:  volumeRatio = VOLUME\_AT\_RATE / Environment.getVolume() # var volumeRatio  Environment.setContribution("Water Level", WATERLEVEL\_RATE \* volumeRatio, MAX\_RATE, True)  Environment.setContribution("Humidity", HUMIDITY\_RATE \* volumeRatio, MAX\_RATE, True)  else:  Environment.setContribution("Water Level", 0, 0, True)  Environment.setContribution("Humidity", 0, 0, True)  if \_\_name\_\_ == "\_\_main\_\_":  setup()  while True:  #loop()  sleep(0) |

14. Siren: -

|  |
| --- |
| from gpio import \*  from time import \*  from ioeclient import \*  from physical import \*  state = 0 # 0 off, 1 on  def main():  setup()  while True:  delay(1000)    def setup():  IoEClient.setup({  "type": "Siren",  "states": [{  "name": "On",  "type": "bool",  "controllable": True  }]  })  global state  add\_event\_detect(0, detect)  IoEClient.onInputReceive(onInputReceiveDone)  state = restoreProperty("state", 0)  setState(state)  def detect():  processData(customRead(0), False)    def onInputReceiveDone(analogInput):  processData(analogInput, True)    def restoreProperty(propertyName, defaultValue):  value = getDeviceProperty(getName(), propertyName)  if not (value is "" or value is None):  if type(defaultValue) is int :  value = int(value)  setDeviceProperty(getName(), propertyName, value)  return value  return defaultValue  def mouseEvent(pressed, x, y, firstPress):  global state  if firstPress:  if state == True:  setState(0)  else:  setState(1)  def processData(data, bIsRemote):  if len(data) <= 0 :  return  setState(int(data))  def setState(newState):  global state  state = newState  if state == True:  digitalWrite(1, HIGH)  else:  digitalWrite(1, LOW)  customWrite(0, state)  IoEClient.reportStates(state)  setDeviceProperty(getName(), "state", state)  if \_\_name\_\_ == "\_\_main\_\_":  main() |

15. Home Speaker: - (main.py)

|  |
| --- |
| from time import \*  from physical import \*  from gpio import \*  from environment import Environment  from ioeclient import IoEClient  from pyjs import \*  VOLUME\_AT\_RATE = 100000  SOUND\_VOLUME = 65  SOUND\_PITCH = 20  SOUND\_WHITE\_NOISE = 20  # Purpose:  # Home Speaker that can play one of several predefined sounds.  g\_sounds = [JsObject({"soundID": 'sound1', "soundPath": '/../Sounds/buzzLow.wav'}),  JsObject({"soundID": 'sound2', "soundPath": '/../Sounds/buzzMedium.wav'}),  JsObject({"soundID": 'sound3', "soundPath": '/../Sounds/buzzHigh.wav'})]  g\_currSound = -1;  # Purpose:  # Setup the Home Speaker. Stop any old sounds and add the sounds to use.  def setup():  global g\_currSound  setDeviceProperty(getName(), 'SOUND\_dB', 0)  IoEClient.setup ({  "type": 'Home Speaker',  "states": [{  "name": 'Signal',  "type": 'number',  "controllable": False  }]  })  destroySounds()  for ind in xrange(0 ,len(g\_sounds)): # var ind  addSound(g\_sounds[ind].soundID, g\_sounds[ind].soundPath)  restoreProperty("SOUND\_dB", 0)  g\_currSound = restoreProperty("Signal", -1)  def restoreProperty (propertyName, defaultValue):  value = getDeviceProperty(getName(), propertyName)  # print "getting:", propertyName, ": ", value  # print "default value:", defaultValue  if not (value is "" or value == "undefined" or value == None) :  if isinstance(defaultValue, (int, float)):  value = int(value)  # print "setting:", propertyName, ": ", value  setDeviceProperty(getName(), propertyName, value)  return value  return defaultValue  # Purpose:  # Update function. Is called once each update.  def loop():  updateState()  delay(1000);  # Purpose:  # Update the sound state, reading from the slot and playing the approperiate sound.  def updateState():  global g\_currSound  playValue = 255\*(analogRead(A0)/1023.0)  if 0 > playValue:  playValue = 0  elif playValue > 255:  playValue = 255    setDeviceProperty(getName(), 'SOUND\_dB', int(playValue \* (60./255)))  if -1 != g\_currSound:  stopSound(g\_sounds[g\_currSound].soundID)  if 0 == playValue:  g\_currSound = -1  else:  g\_currSound = int((playValue-1) / (255./len(g\_sounds)));  if -1 is not g\_currSound:  playSound(g\_sounds[g\_currSound].soundID, 1)  IoEClient.reportStates(g\_currSound)  setDeviceProperty(getName(), "Signal", g\_currSound)  updateEnvironment()  def updateEnvironment():  volumeRatio = VOLUME\_AT\_RATE / Environment.getVolume()  if g\_currSound >= 0:  Environment.setContribution("Sound Volume", SOUND\_VOLUME\*volumeRatio, SOUND\_VOLUME\*volumeRatio, False)  Environment.setContribution("Sound Pitch", SOUND\_PITCH\*(g\_currSound+1), SOUND\_PITCH\*(g\_currSound+1), False)  Environment.setContribution("White Noise", SOUND\_WHITE\_NOISE, SOUND\_WHITE\_NOISE, False)  else:  Environment.setContribution("Sound Volume", 0, 0, False)  Environment.setContribution("Sound Pitch", 0, 0, False)  Environment.setContribution("White Noise", 0, 0, False)  if \_\_name\_\_ == "\_\_main\_\_":  setup()  while True:  loop()  sleep(0) |

Pyjs.py

|  |
| --- |
| ##############################################################  # Helper class to help porting code using JS object notation:  #  # JS: var obj = {type: "unknown"}  # JS: obj.type = "analog"  #  # PY: obj = JsObject({"type" : "unknown"})  # PY: obj.type = "analog"  #  class JsObject(dict):  def \_\_init\_\_(self, d):  for k in d.keys():  setattr(self, k, d[k])    ##############################################################  # Map value from one range to another  #  def js\_map(x, inMin, inMax, outMin, outMax):  return (x - inMin) \* (outMax - outMin) / (inMax - inMin) + outMin |

16 Street Lamp(main.py)

|  |
| --- |
| from time import \*  from physical import \*  from gpio import \*  from ioeclient import IoEClient  from pyjs import JsObject  from udp import \*  from sensor\_light import SensorLight  from sensor\_motion import SensorMotion  DELAY = 250  SIZE = JsObject({ # var SIZE  "width": 150,  "height": 150  }) # largest component size  SERVER\_PORT = 1234 # var SERVER\_PORT  SERVER\_IP = "192.168.0.100" # var SERVER\_IP  socket = None # var socket  sensors = None # var sensors  def setup ():  global socket, sensors    IoEClient.setup({  "type": "Street Lamp",  "states": [{  "name": "Light",  "type": "number",  "controllable": False  }, {  "name": "Light gradient",  "type": "options",  "options": {  "-1": "Decreasing",  "0": "No Change",  "1": "Increasing"  },  "controllable": False  }, {  "name": "Motion",  "type": "number",  "controllable": False  }, {  "name": "Moton gradient",  "type": "options",  "options": {  "-1": "Decreasing",  "0": "No Change",  "1": "Increasing"  },  "controllable": False  }]  })    sensors = JsObject({  "light": SensorLight(),  "motion": SensorMotion()  })    socket = UDPSocket()  socket.begin(SERVER\_PORT)  def loop ():  global dataPrev  xpos = getX() # var xpos  ypos = getY()  devices = devicesAt(xpos, ypos, SIZE.width, SIZE.height \* 2) # var devices  sensors.light.update(sensors, devices)  sensors.motion.update(sensors, devices)    sendData()  sendReport()  # send data to the server  def sendData():  data = "".join([  "streetlamp," , str( getSerialNumber() ) , "^",  "light," , str( sensors.light.value() ) , ",",  str( sensors.light.valueDirection() ) , "^",  "motion," , str( sensors.motion.value() ) , ",",  str(sensors.motion.valueDirection() )  ])  socket.send(SERVER\_IP, SERVER\_PORT, data)  # send report to reg server  def sendReport ():  report = ",".join([  str( sensors.light.value() ),  str( sensors.light.valueDirection() ),  str( sensors.motion.value() ),  str( sensors.motion.valueDirection() )  ])  IoEClient.reportStates(report)  if \_\_name\_\_ == "\_\_main\_\_":  setup()  while True:  loop()  delay(DELAY) |

Pyjs.py

|  |
| --- |
| ##############################################################  # Helper class to help porting code using JS object notation:  #  # JS: var obj = {type: "unknown"}  # JS: obj.type = "analog"  #  # PY: obj = JsObject({"type" : "unknown"})  # PY: obj.type = "analog"  #  class JsObject(dict):  def \_\_init\_\_(self, d):  for k in d.keys():  setattr(self, k, d[k])    ##############################################################  # Map value from one range to another  #  def js\_map(x, inMin, inMax, outMin, outMax):  return (x - inMin) \* (outMax - outMin) / (inMax - inMin) + outMin |

Sensor\_light.py

|  |
| --- |
| from time import \*  from physical import \*  from gpio import \*  from environment import Environment  from pyjs import \*  # cl-sensor-light.js  # Env.Sunlight is assumed to be from 0 to 100%  class SensorLight:  def \_\_init\_\_(self): # var SensorLight  self.eLightMin = 0.  self.eLightMax = 100. # this is %, per Environment semantics for sunlight  self.eLightMinValueOn = self.eLightMin + (self.eLightMax - self.eLightMin) / 3.  # init  setComponentOpacity("SensorLightOff", 0)  setComponentOpacity("SensorLightOn", 0)  setComponentOpacity("SensorLight", 0)  self.elight = Environment.get("Sunlight")  self.elightPrev = None  self.sync\_to\_env()  self.update\_visuals()  def update(self, sensors, devices):  self.sync\_to\_env()  self.update\_visuals()  def valueDirection(self):  if self.elightPrev is self.elight:  return 0  elif self.elightPrev < self.elight:  return 1  return -1  def value(self):  return self.elight  # private  def update\_visuals(self):  # value  opacity = 0 # var opacity  value = self.value()  if value < self.eLightMinValueOn:  opacity = 1 - (value - self.eLightMin) / (self.eLightMinValueOn - self.eLightMin)  setComponentOpacity("SensorLightOn", 1)  setComponentOpacity("SensorLightOff", 0)  else:  setComponentOpacity("SensorLightOn", 0)  setComponentOpacity("SensorLightOff", 1)  setComponentOpacity("SensorLightLevel", opacity)  # private  def sync\_to\_env(self):  self.elightPrev = self.elight  self.elight = Environment.get("Sunlight")  if self.elight < self.eLightMin:  self.elight = self.eLightMin  if self.elight > self.eLightMax:  self.elight = self.eLightMax |

Sensor motion.py

|  |
| --- |
| from time import \*  from physical import \*  from gpio import \*  from pyjs import \*  # cl-sensor-motion.js  class SensorMotion:  def \_\_init\_\_(self): # var SensorMotion  self.motionPrev = 0  self.motion = 0  self.devices = {}  # init  self.update\_visuals()  def update(self, sensors, devices):  self.update\_motion(sensors, devices)  self.update\_visuals()  def valueDirection(self):  if self.motionPrev is self.motion:  return 0  elif self.motionPrev < self.motion:  return 1  return -1  def value(self):  if self.motion:  return self.motion  elif self.valueDirection() is not 0:  return 1  return 0  # private  def update\_visuals(self):  # value  value = self.value() # var value  if value > 0:  setComponentOpacity("SensorMotionOn", 1)  setComponentOpacity("SensorMotionOff", 0)  else:  setComponentOpacity("SensorMotionOn", 0)  setComponentOpacity("SensorMotionOff", 1)  # private  def update\_motion(self, sensors, devices):  myname = getName() # var myname  self.motionPrev = self.motion  for n in self.devices: # var n  self.devices[n].current = False  for i in xrange(0, len(devices)) :  name = devices[i] # var name  mydev = self.devices.get(name)  if name is myname:  continue  if mydev is None:  self.devices[name] = JsObject({  "current": True  })    self.motion += 1  else:  mydev.current = True  for nn in self.devices: # var nn  if not self.devices[nn].current:  del self.devices[nn]  self.motion -= 1 |

17. Old Car

|  |
| --- |
| from time import \*  from physical import \*  from gpio import \*  from environment import Environment  from ioeclient import IoEClient  CO\_RATE = 1./3600 # 1% per hour  CO2\_RATE = 2./3600  SMOKE\_RATE = 3./3600  TEMPERATURE\_RATE = 1./3600  VOLUME\_AT\_RATE = 100000.  MAX\_RATE = 1.e6  state = 0  def updateEnvironment ():  if state == 1 :  volumeRatio = VOLUME\_AT\_RATE / Environment.getVolume()  Environment.setContribution("CO", CO\_RATE\*volumeRatio, MAX\_RATE, True)  Environment.setContribution("CO2", CO2\_RATE\*volumeRatio, MAX\_RATE, True)  Environment.setContribution("Smoke", SMOKE\_RATE\*volumeRatio, MAX\_RATE, True)  Environment.setContribution("Ambient Temperature",TEMPERATURE\_RATE\*volumeRatio, MAX\_RATE, True)  else:  Environment.setContribution("CO", 0, 0, True)  Environment.setContribution("CO2", 0, 0, True)  Environment.setContribution("Smoke", 0, 0, True)  Environment.setContribution("Ambient Temperature", 0 , 0, True)  def setup ():  global state  state = restoreProperty("state", 0);  setState(state)  def restoreProperty(propertyName, defaultValue):  value = getDeviceProperty(getName(), propertyName)  if not (value is "" or value is None):  if type(defaultValue) is int :  value = int(value)  setDeviceProperty(getName(), propertyName, value)  return value  return defaultValue  def mouseEvent (pressed, x, y, firstPress):  if firstPress:  setState(0 if state else 1)  def setState (newState):  global state  if newState == 0 :  digitalWrite(1, LOW)  else:  digitalWrite(1, HIGH)  state = newState  setDeviceProperty(getName(), "state", state)  updateEnvironment()  if \_\_name\_\_ == "\_\_main\_\_":  setup()  while True:  sleep(3600) |

18. Garage Door

|  |
| --- |
| from time import \*  from physical import \*  from gpio import \*  from environment import Environment  from ioeclient import IoEClient  ENVIRONMENTS = [  "Argon", "CO", "CO2", "Hydrogen", "Helium", "Methane",  "Nitrogen", "O2", "Ozone", "Propane", "Smoke"]  ENVIRONMENT\_MAX\_IMPACT = -0.04 # 4% max when door opens  TEMPERATURE\_TRANSFERENCE\_MULTIPLIER = 1.50 # increase speed 25% when door open  HUMIDITY\_TRANSFERENCE\_MULTIPLIER = 1.50  GASES\_TRANSFERENCE\_MULTIPLIER = 2  state = 0 # 0 off, 1 on  def on\_event\_detect\_0 () :  processData(customRead(0), False)  def on\_input\_receive(input) :  processData(input, True)  def setup ():  global state  IoEClient.setup ({  "type": "Garage Door",  "states": [{  "name": "On",  "type": "bool",  "controllable": True  }]  })  IoEClient.onInputReceive(on\_input\_receive)  add\_event\_detect(0, on\_event\_detect\_0)  state = restoreProperty("state", 0);  setState(state)  def restoreProperty (propertyName, defaultValue):  value = getDeviceProperty(getName(), propertyName)  if value :  if isinstance(defaultValue, (int, float)):  value = int(value)  setDeviceProperty(getName(), propertyName, value)  return value  return defaultValue  def mouseEvent (pressed, x, y, firstPress):  if firstPress :  setState(0 if state else 1)  def updateEnvironment ():  if state == 1:  for e in ENVIRONMENTS:  emax = Environment.get(e) \* ENVIRONMENT\_MAX\_IMPACT  # the emax is reached in an hour, so we divide by 3600 to get seconds  # then this rate is also based on 100,000 cubic meters (approx. coporate office size)  rate = emax / 3600 \* 100000 / Environment.getVolume()  Environment.setContribution(e, rate, emax, True)  Environment.setTransferenceMultiplier(e, GASES\_TRANSFERENCE\_MULTIPLIER)  Environment.setTransferenceMultiplier("Ambient Temperature", TEMPERATURE\_TRANSFERENCE\_MULTIPLIER)  Environment.setTransferenceMultiplier("Humidity", HUMIDITY\_TRANSFERENCE\_MULTIPLIER)  else:  for e in ENVIRONMENTS:  Environment.setContribution(e, 0, 0, True)  Environment.removeCumulativeContribution(e)  Environment.setTransferenceMultiplier(e, 1)  Environment.setTransferenceMultiplier("Ambient Temperature", 1)  Environment.setTransferenceMultiplier("Humidity", 1)  def processData (data, bIsRemote):  if data is None or data is "":  return  setState(int(data))  def setState (newState):  global state  state = newState  digitalWrite(1, HIGH if state else LOW)  customWrite(0, state)  IoEClient.reportStates(state)  setDeviceProperty(getName(), "state", state)  updateEnvironment()  if \_\_name\_\_ == "\_\_main\_\_":  setup()  while True:  sleep(0) |