ONOS Documentation

Release

Author

May 25, 2017

Contents

Contents:

CHAPTER 1

arduino_handler module

CHAPTER 2

arduinoserial module

CHAPTER 3

conf module

This module is used to import all the configurations from the saved files at the onos startup

conf.**importConfig**()
: This function imports all the data and configurations from the files saved on storage memory.|br| The file are located in the config_files directory

conf.**newDefaultWebObj**(*name*)
: Return a new web_object given only its name, used to create new web_objects for exemple when a new zone is created

conf.**newDefaultWebObjBody**(*name*)
: Return a new web_object given only its name, used to create the zone html body object

conf.**newNodeWebObj**(*name*, *objType*, *node_sn*, *pinList=[]*)
: Return a new web_object given its name,objType,node_sn,pinList used to create new web_objects for exemple when a new node is added

conf.**readConfigurationsFromSavedFile**(*key*)
: Given a key it reads the value in the json dictionary from a file config_files/cfg.json saved on the storage memory

conf.**readDictionaryFromSavedFile**(*key*)
: Given a key it reads the value in the json dictionary from a file config_files/data.json saved on the storage memory

CHAPTER 4

globalVar module

This module is imported from all the others and it stores the global variables and some global functions.

globalVar.**getErrorTimeString**()
Called when an error occours ,return the current time and a progressive number of the error, incrementing the error_count. Used to send the error time and error_count to debug the software.

globalVar.**getListPinsConfigByHardwareModel**(*hwName*, *pin_mode*)
Given hardware name and a pin mode , return a list containing the pin numbers that are configurated with that pin_mode

Parameters

- **hwName** – The type of the hardware node for examples:
  - ProminiA
  - Plug6way
  - RouterGL
  - RouterGA
  - RouterRB
  - Any other hardware type added as hardwareModelDict key in globalVar.py
- **pin_mode** – The pin mode you are searching for:
  - **sr_relay:** Latch relay where the first pin in the tuple is set and the second is reset
  - **digital_output:** Output digital pins
  - **digital_input :** Input digital pins
  - **analog_input** [] Analog input
  - **servo_output :** Servomotor control pin
  - **analog_output** [] Pwm output pin

globalVar.**getListUsedPinsByHardwareModel**(*hwName*)
Given an hardware name return a list with all the pins used by a hardware model

Parameters **hwName** – The type of the harware node for examples:

- ProminiA
- Plug6way
- RouterGL
- RouterGA
- RouterRB
- Any other hardware type added as hardwareModelDict key in globalVar.py

globalVar.**get_ip_address**()

globalVar.**logprint**(*message*, *verbose=0*, *error_tuple=None*)
|Print the message passed and if the system is in debug mode or if the error is important send a mail |Remember, to clear syslog you could use : > /var/log/syslog |To read system log in openwrt type:logread

CHAPTER 5

hw_node module

This module is responsable for the handling of each node in the system.
It generates the setup pin configuration to send to the nodes.
It stores each node pin setup and last time sync.

class hw_node.**HwNode**(*NodeSerialNumber*, *hwModel*, *address*, *node_fw*)

close()

getHwType()
Return the node hardware type like:

- gl.inet_only
- arduino_promini
- rasberry_b_rev2_only
- arduino2009

getLastNodeSync()
Return the the time from last node time sync

getMaxPinNumber()
Return the number of pin present in the node

getNodeActivity()
Return the node activity status.

getNodeAddress()
Return the node address , if the address is 0 then the node is the arduino over usb (todo)

getNodeFwVersion()

Return a string containing the node firmware version

An example is “5.14”

getNodeHwModel()

getNodeObjectAddress(*objectName*)

Given a objectname it will return its address in the node

getNodeObjectFromAddress(*objectAddress*)

Get the objectname in the node address

getNodeSectionMode()

Warning Never used

Return a list containing the node mode for each section (8 bit)
(the section0 is relative to the first 8 pins , from pin 0 to pin 7) msb left
(the section1 is relative to the pins from 8 to 15)
(the section2 is relative to the pins from 16 to 23)
(the section3 is relative to the pins from 24 to 31)
(the section4 is relative to the pins from 32 to 39)
(the section5 is relative to the pins from 40 to 47)
(the section6 is relative to the pins from 48 to 55)

getNodeSectionModeByPin(*pin*)

Deprecated , not used anymore. Given a pin return the node mode for the section (8 bit) containing that pin.

getNodeSectionStatusByPin(*pin*)
Given a pin return a tuple with:

- The number of the section where the pin is located
- The the node status register for the section (8 bit) containing that pin

Structure:
(the section0 is relative to the first 8 pins , from pin 0 to pin 7) msb left
(the section1 is relative to the pins from 8 to 15)
(the section2 is relative to the pins from 16 to 23)
(the section3 is relative to the pins from 24 to 31)
(the section4 is relative to the pins from 32 to 39)
(the section5 is relative to the pins from 40 to 47)
(the section6 is relative to the pins from 48 to 55)

getNodeSerialNumber()

Return the node serial number.
For example “Plug6way0001”

getNodeStatusList()

Warning Never used

Return a list containing the node status for each pin.

getNodeTimeout()

Return the time after which the node is declared inactive.
So if getLastNodeSync() is greater than this self.timeout the node will be setted as inactive
self.timeout is readed from hardwareModelDict in globalVar.py

getPinMode(*pin*)
Deprecated , not used anymore

getPinModeList()

Warning Never used

Return the Pin mode list containing all the pin mode.

getPinStatus(*pin*)
Deprecated , not used anymore

getSetupMsg()
Return a encoded string containing the setup mode for the node pins

I need to encode the pin setup in a sinmple and compact way. Here the protocol:

```
start with 's='
then  add    9 bytes that rappresent the pin used (1 for pin used 0 for not␣
↪used)
after that   9 bytes that rappresent the digital pin setup from pin0 to pin␣
↪127
after that   9 bytes that tell arduino which pin to set as analog input
after that   9 bytes that tell arduino which pin to set as pwm output
after that   9 bytes that tell arduino which pin to set as servo output
then  1   byte for future use , for now '#'
example:"s=000000000000000000000000000000000000000000000000#"
the 0 are not 0 but the value corrisponding to ascii '0'
total 48 byte
```

getUsedPins()
Return the list of used pins

getnodeObjectsDict()

Get the objectname in the node address

isPinOk(*pin*)
Return 1 if the given pin exist in this hardware node, 0 otherwise

setAnalogPinInputStatus(*pin*, *status*)
Deprecated , not used anymore.

setDigitalPinInputStatus(*pin*, *status*)
Deprecated , not used anymore.

setDigitalPinOutputStatus(*pin*, *status*)
Deprecated , not used anymore

setNodeActivity(*value*)
Set the node activity status with the one given.

Parameters **value** –

- **The value to set the node activity should be a integer of 0 or 1**

0 If the node is inactive
1 If the node is active

setNodeAddress(*address*)

Set the node address with the string passed
Example: "192.168.101.10"

setNodeAnalogInputStatusFromReg(*pin_number*, *low_byte*, *high_byte*)

Given a pin number, and two bytes return the analog value in an single integer.
Since arduino analogRead return a 10 bit analog value to send it I need to split it in two bytes (8 bit each) so arduino will send 2 bytes the low_byte and the high_byte , this function will rebuild the number from those two bytes

setNodeFwVersion(*nodeVersion*)

Set the node firmware version with the given string
Example "4.15"

setNodeObjectAddress(*objectAddress*, *objectName*)

Set the objectname to an address in the node

setNodePinMode(*pin*, *mode*)

Given a pin number and a mode, set the pin mode
The options for mode are:

•"DOUTPUT" : digital output
•"AOUTPUT" : analog output
•"SOUTPUT" : servo motor output
•"DINPUT" : digital input
•"AINPUT" : analog input

setNodeSectionDInputStatus(*section_number*, *status_byte*)

Set the node status pins of a section (8 bit) received from arduino.
if the section status is different from the previous one then check
what pins changed and ask the webserver.py to change the webobj status of the relative pins

i don't need a setNodeSectionDoutputStatus because the output status will be saved in the webobject status.

updateLastNodeSync(*time*)
When called update the time from last node time sync with the given one

CHAPTER 6

mail_agent module

This module handles the onos email system.
It handles the incoming mails and the outbound email.

Warning:

You musn't have the fowarding from the mail you use as onos income mail.
You musn't use a mail programm to automaticly download the mail.

Because otherwise the mail will be donwloaded elsewhere and onos will not be able to read it since it reads only the unreaded mails.

`mail_agent.`**`get_text`**`(`*`msg`*`)`

Given the python mail object, parses email message text and return the mail text content.
This doesn't support infinite recursive parts, but mail is usually not so naughty.

`mail_agent.`**`receiveMail`**`(`*`mail_conf`*`, `*`imaplib`*`, `*`email`*`)`

This function connect to the mail server and download all the unread mails.
Then if a mail contain the "onos=" string, the mail is added to a list which will be returned.
Otherwise the mail is setted as unreaded in the mail server since is not a mail command for onos.
To connect to the mail server the credential contained in mail_conf[] dictionary will be used.

Parameters

- **`mail_conf`** – The dictionary containing the mail credential,server address and smtp_port. (for now i tried only with gmail).

- **imaplib** – The imaplib library imported in globalVar.py
- **email** – The email library imported in globalVar.py

mail_agent.**sendMail**(*receiver_user_mail*, *mailtext*, *mailSubject*, *mail_conf*, *smtplib*, *string*)

This function send a mail from the onos system.

> **Warning:** If you want to send a mail, is better to add it to the mailQueue using: mailQueue.put({"mail_address":m_sender,"mailText":mailText,"mailSubject":mailSubject}) In this way the mail will be sent after the previous ones are sent.

Parameters

- **receiver_user_mail** – The mail receiver
- **mailtext** – The mail text content
- **mailSubject** – The mail subject
- **mail_conf** – The dictionary containing the mail credential,server address and smtp_port. (for now i tried only with gmail).
- **imaplib** – The imaplib library imported in globalVar.py
- **string** – The string library imported in globalVar.py

CHAPTER 7

node_query_handler module

This Modules handles all the query to the nodes.
it will receive the data from the router_handler and it will send the query to the nodes.
It will also retry sending the message if the node doesn't answer right or if doesn't answer at all.
If the nodes confirm the command was received then this module will tell onos to set the web object status
to reflect the new node status after the command was received.

node_query_handler.**handle_new_query_to_network_node_thread**()

This is a thread function that will run until every request in the queryToNetworkNodeQueue is done.
It will get each query from queryToNetworkNodeQueue and call make_query_to_network_node()
While the query is running the current_node_handler_list will contain the node serialnumber being queried
In this way onos will avoid to make multiple simultaneos query to the same node.

node_query_handler.**handle_new_query_to_radio_node_thread**(*serialCom*)

This is a thread function that will run until every request in the queryToRadioNodeQueue is done.
It will get each query from queryToRadioNodeQueue and call make_query_to_radio_node()
While the query is running the current_node_handler_list will contain the node serialnumber being queried
In this way onos will avoid to make multiple simultaneos query to the same node.

node_query_handler.**make_query_to_network_node**(*node_serial_number*, *query*, *objName*, *status_to_set*, *user*, *priority*, *mail_report_list*)

This function make a query to a powerline/ethernet node and wait the answer from the node.
If the answer is positive

it will add to the priorityCmdQueue the command to change the web_object status
from pending to succesfully changed .

If the answer from the node is an error or the node is not responding
the query will be repeated x times before giving up.

node_query_handler.**make_query_to_radio_node**(*serialCom*, *node_serial_number*, *query*, *number_of_retry_already_done*)

This function make a query to a radio/serial node and wait the answer from the serial gateway.
If the answer is positive
it will add to the priorityCmdQueue the command to change the web_object status
from pending to succesfully changed .
If the answer from the node is an error or the node is not responding
the query will be repeated x times before giving up.

CHAPTER 8

pcduino module

CHAPTER 9

router_handler module

This Modules handles all the comunication to nodes.
It will routes the varius commands knowing where to send them based on the node address.

If the node_address of the given receiver is 0 the data will be written to the local OnosCenter pins.

if the node_address of the given receiver is 1 the data will be sent to the OnosCenter serial port to an arduino (not implemented yet)

If the node_address of the given receiver is between 2 to 255 :

> OnosCenter will send the command to write the data to an arduino wich will then transmit it to the desired wireless node.(not implemented yet)

If the node_address of the given receiver is a string greater than 6 characters: The data will be sent over ethernet commmunication.

class router_handler.**RouterHandler**(*hardwareModelDict*, *router_sn*)

close()

composeChangeNodeOutputPinStatusQuery(*pinNumbers*, *node_obj*, *objName*, *status_to_set*, *node_serial_number*, *node_address*, *out_type*, *user*, *priority*, *mail_report_list*)

> Compose the correct query to change an output pin status on a remote node.
> The examples are:
>
> WARNING old documentation... todo update it

"onos_r"+pin0+pin1+"v"+status_to_set+"s"+node_serial_number+"_#]" to set a sr_relay to status_to_set (0 or 1)

onos_r0607v1sProminiS0001f000_#] pin6 and 7 used to control a s/r relay and turn it to set

"onos_d"+pin_number+"v"+"00"+status_to_set+"s"+node_serial_number+"_#]" to set a digital pin to 0 or 1

onos_d07v001sProminiS0001f000_#] set digital pin7 to 1

"onos_s"+pin_number+"v"+status_to_set+"s"+node_serial_number+"_#]" to set a servopin from 0 to 180

onos_s07v180sProminiS0001f000_#] set servo pin7 to 180

"onos_a"+pin_number+"v"+status_to_set+"s"+node_serial_number+"_#]" to set a analogpin from 0 to 255

onos_a05v100sProminiS0001f000_#] set analog pin5 to 100

The node_serial_number is a hexadecimal number refering to the node serial number

The node_address is a 3 byte ashii that rappresend the wireless node address(if the node is wireless,otherwise is ignored)

the value must be between 0 and 255 it will be setted to "999" if is not used..

connectSerialPort()

getProgressive_msg_id()

Get a progressive id number to append it to the message in order to make it unique

getRouterName()

initializeArduinoCommunication()

outputWrite(*node_serial_number*, *pinList*, *statusList*, *node_obj*, *objName*, *previous_status*, *statusToSetWebObject*, *output_type*, *user*, *priority*, *mail_report_list*)

read_router_pins()

Thread function to read changing of pin status on the embedded linux board (glinet,raspberry ecc)

searchObjectBaseName(*obj_selected*, *node_serial_number*)

setAddressToNode(*node_serial_number*, *new_address*)

setHwPinMode(*node_address*, *pinNumber*, *mode*)

writeRawMsgToNode(*node_serial_number*, *node_address*, *msg*)

CHAPTER 10

time_zone module

This module will set the right timezone to the openwrt system calling:
‘ntpd -q -p 0.openwrt.pool.ntp.org’
And executing ‘echo “export TZ=’+timezone+”’>> /etc/profile’

CHAPTER 11

web_object module

class web_object.**WebObject**(*name, obj_type='b', start_status=0, styleDictionary={}, htmlDictionary={}, cmdDictionary={}, note=' ', hardware_pin=[9999], HwNodeSerialNumber=0, spareDict={}*)

Bases: object

HwNodeSerialNumber

InitFunction()

addToGroup(*username*)

attachScenario(*scenario*)

baz

changeCommand(*status*, *cmd*)

changeCommand0(*f0*)

changeCommand1(*f1*)

checkPermissions(*user*, *action*, *priority*)

checkRequiredPriority(*agent_priority*)

cmdDict

getAttachedPinList()

Return a list containing the object pins used in the hardware and if the webobject is a is a digital_obj or an analog_obj or ...obj it will return the address of this object in the node,
example: 0 means first object in the node, 1 means second object in the node ...

getCommand0()

getCommand1()

getGroup()

getHtml()

getHtml0()

getHtml1()

getHtmlDict()

getHwNodeSerialNumber()

getInitCommand()

getListAttachedScenarios()

getMailReport()

getName()

getNotes()

getObjActivity()

getObjectDictionary()

getOtherHtml()

getOtherStyle()

getOwner()

getPermissions()

getPreviousStatus()

getPreviousStatusForScenario()

getRequiredPriority()

getStartStatus()

getStatus()

getStatusForScenario()

getStyle()

getStyle0()

getStyle1()

getType()

hardware_pin

htmlDict

name

note

object_type

removeAttachedScenario(*scenario*)

removeFromGroup(*username*)

replaceAttachedScenario(*previous_scenario_name*, *new_scenario_name*)

setAttachedPin(*pin*)

setCommand0(*command0*)

setCommand1(*command1*)

setHtml0(*html0*)

setHtml1(*html1*)

setHtmlDict(*htmlDict*)

setHtmlDictValue(*key*, *value*)

setHtmlWait(*html_w*)

setHwNodeSerialNumber(*HwNodeSerialNumber*)

setInitCommand(*init_command*)

setMailReport(*mail_to_add_to_list*)

setName(*new_name*)

setNotes(*note*)

setObjActivity(*value*)

setOwner(*owner*)

setPermissions(*perm*)

setRequiredPriority(*priority*)

setStatus(*status*)

setStyle0(*style*)

setStyle1(*style1*)

setType(*current_type*)

spareDict

start_status

styleDict

validateStatusToSetObj(*status*)

CHAPTER 12

webserver module

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

H

I

L

M

N

O

R

S

T

U

V

W