

collection of use cases



Disclaimer

This page is originally from a git repository where its content can be managed and presented in a more dynamical way. This is a snapshot as of today (05 Jan 2023). There will be no frequent updates or alike. Really new content comes with a new documentation platform – some day! Contact o2a-support@awi.de for further assistance, comments, complaints or praise

- 1 Use Case 1: user and items thing
 - 1.1 Use case 1.1: Which item is associated with which user?
 - 1.2 Use case 1.2: Which items have the user x with a certain role?
- 2 Use Case 2: write a certain event (e.g. info about calibration) to item xy
- 3 Use Case 3: create a list of all items of a platform that measure parameter xyz
- 4 Use Case 4: adding/removing/modifying a parameter to item xyz
- 5 Use Case 5: create a list for my item that holds necessary infos, such as 'parameter name', 'parameter type', 'unit' and comments', for a PANGAEA data publication.
- 6 Use Case 6: adding/removing a user with a certain role to an item
 - 6.1 add
 - 6.2 remove
- 7 Use Case 7: show all items (e.g. ctd) that measure a certain parameter type
- 8 Use Case 8: Which item types are available and how are they described
- 9 Use Case 9: explore relevant, human-readable metadata fields of an item
 - 9.1 compile parameter list
 - 9.1.1 approach 1
 - 9.1.2 approach 2
 - 9.2 make a list of all subitems
- 10 Use Case 10: create a new item
- 11 Use Case 11: reassign items

Use Case 1: user and items thing

Use case 1.1: Which item is associated with which user?

- [X] PY
- [X] R

input:

- user email **or**
- user firstname and lastname

Per API: Search for all items of contact x, i) by (system) email address, ii) by contact names.

1. get /sensors/device /getItemsOfContact/{contactEmail} **OR**
2. get /sensors/contacts /getContactsByName/{firstName}/{lastName}
--> email address will be output, then go on with step 1

Use case 1.2: Which items have the user x with a certain role?

- [X] PY
- [X] R

input:

- input from 1.1
- user role

1. von use case 1.1

use case 1 Python

```
'''
basic documentation of sensor.awi.de REST-API can be found here:
https://sensor.awi.de/api/
'''
## necessary libraries

import json
import requests
import datetime

## PROD
#sensorURL<- "https://sensor.awi.de/rest/sensors/"
## SANDBOX
sensorURL <- "https://sandbox.sensor.awi.de/rest/sensors/"

## + ===== | xxx |
===== + ##
'''
USE CASE 1: We are looking for a list of items that have a certain user in
their contact list.
1.1.a We know the contact email address (as in sensor.awi.de) of the user.
'''
## + ===== | xxx |
===== + ##

email = 'marcel.nicolaus@awi.de'

## create link for API call
link = sensorURL + "device/getItemsOfContact/" + email

## calling api
answer = requests.get(link)
## the content itself, read as a json object
fullList = json.loads(answer.content)

## status of request (see httpstatuses.com/)
```

- a. get /sensors
 /device
 /getItemsOfContact/
 {contactEmail} OR
- b. get /sensors
 /contacts
 /getContactsByName/{firstName}/
 {lastName} -->
 email address as
 output then go to
 step 1
2. get /sensors/contacts
 /getAllContactRoles -->
 pick out which ID is related to
 which role type (e.g. *Data
 scientist*="id": 29]
3. filter (role) ID-based
 - PI==True --> put to
 list
 - PI==False --> go on
 - ...

```

answer.status_code

## since the result can be 0 items, 1 item, and >1 items , we filter the
item ids according the number of items and write to a list
if len(fullList) == 0:
    itemList = []
elif len(fullList) == 1:
    itemList = [fullList[0]['id']]
else:
    itemList = [i['id'] for i in fullList]

## there we go:
print(itemList)

## + ===== | xxx |
+ ##
'''

USE CASE 1: We are looking for a list of items that have a certain user in
their contact list.
1.1.b We know the last name (and if necessary the first name) of the user.
Aim: get a single email address.
'''

## + ===== | xxx |
+ ##

## creation of the API call
link = sensorURL + 'contacts/getAllSystemContacts'

## CASE: unique last name or no contact found

## variables
lastName = 'Gerchow'

## calling api
answer = requests.get(link)
x = json.loads(answer.content)

## loop over entire list
for a in x:
    if a['lastName'] == lastName:
        print(a['email'])

## CASE: non-unique last name or no contact

## variables
lastName = 'Nicolaus'
firstName = 'Rita'

## calling api
answer = requests.get(link)
x = json.loads(answer.content)

## loop over entire list
for a in x:
    if a['lastName'] == lastName and a['firstName'] == firstName:
        print(a['email'])

## + ===== | xxx |
+ ##
'''

USE CASE 1: We are looking for a list of items that have a certain user in
their contact list.
1.2. We have info from 1.1 but need that user only with a certain contact
role (e.g. 'Principal Investigator').
'''

## + ===== | xxx |
+ ##

## which user should be searched?

```

```

## remember 'email' from 1.1?
#email = 'sandra.tippenhauer@awi.de'

## what role should be filtered?
inputRole = 'Principal Investigator'
inputRole = 'Editor'

## create empty lists
lstID, lstURN = [], []

## recycling of itemList from 1.1
for item in itemList:
    link = sensorURL + "contacts/getDeviceContacts/" + str(item)
    answer = requests.get(link)
    x = json.loads(answer.content)

    ## create dictionaries to map uuids to vocables
    dictRoles = {}
    dictAccounts = {}
    for i in x:
        if isinstance(i['contact'], dict):
            dictAccounts[i['contact']['@uuid']] = i['contact']['email']
        if isinstance(i['vocable'], dict):
            dictRoles[i['vocable']['@uuid']] = i['vocable']['generalName']

    ## extracting from item
    user, role = [], []
    for i in x:
        if isinstance(i['vocable'], dict):
            role.append(i['vocable']['@uuid'])
        elif isinstance(i['vocable'], str):
            role.append(i['vocable'])
        else:
            pass
        ##
        if isinstance(i['contact'], dict):
            user.append(i['contact']['@uuid'])
        elif isinstance(i['contact'], str):
            user.append(i['contact'])
        else:
            pass

    ## merging info from item and mapping dict
    for i in range(len(user)): ## <-- could improve!!!
        if dictRoles[role[i]] == inputRole and dictAccounts[user[i]] ==
email:
            lstID.append(item)
            lstURN.append(x[0]['item']['urn'])

## neat output as some sort of report ...
print(json.dumps({'user':email,
                  'timeOfQuery':datetime.datetime.isoformat(datetime.
datetime.utcnow()),
                  'role':inputRole,
                  'urn':lstURN,
                  'id':lstID}
))

## ... or just as a single list if item IDs
print(lstID)

## eof

```

use case 1 R

```

## '''
## basic documentation of sensor.awi.de REST-API can be found here:
https://sensor.awi.de/api/

```

```

## '''
settingUpThings <- function(x){
  mirror <- 'https://ftp.gwdg.de/pub/misc/cran'
  newPackages <- x[!(x %in% installed.packages()[, 'Package'])]
  if(length(newPackages)){
    install.packages(newPackages, dep = TRUE, repos = mirror)
  }
  lapply(x, library, character.only = TRUE, logical.return = TRUE)
}

libsToBeLoaded <- c('jsonlite', 'plyr', 'lubridate', 'httr')
settingUpThings(libsToBeLoaded)

## PROD
#sensorURL<- "https://sensor.awi.de/rest/sensors/"
## SANDBOX
sensorURL <- "https://sandbox.sensor.awi.de/rest/sensors/"

## + ===== | xxx |
## + ===== + ##
## '''
## USE CASE 1: We are looking for a list of items that have a certain user
## in their contact list.
## 1.1.a We know the contact email address (as in sensor.awi.de) of the
## user.
## '''
## + ===== | xxx |
## + ===== + ##

email <- 'marcel.nicolaus@awi.de'

## create link for API call
link <- paste0(sensorURL, "device/getItemsOfContact/", email)

## calling api
answer <- GET(link)
## the content itself, read as a json object
fullList <- content(answer, as = 'parsed')

## status of request (see httpstatuses.com/)
answer$status_code

## since the result can be 0 items, 1 item, and >1 items , we filter the
## item ids according the number of items and write to a list
if (length(fullList) == 0){
  itemList <- list()
} else if (length(fullList) == 1){
  itemList <- list(fullList[[1]]$id)
} else {
  itemList <- list()
  for (i in seq_along(fullList)){
    itemList[[i]] <- fullList[[i]]$id
  }
}

## there we go:
print(itemList)

## + ===== | xxx |
## + ===== + ##
## '''
## USE CASE 1: We are looking for a list of items that have a certain user
## in their contact list.
## 1.1.b We know the last name (and if necessary the first name) of the
## user. Aim: get a single email address.
## '''
## + ===== | xxx |
## + ===== + ##

```

```

## creation of the API call
link <- paste0(sensorURL,'contacts/getAllSystemContacts')

## CASE: unique last name or no contact found

## variables
lastName <- 'Gerchow'

## calling api
answer <- GET(link)
x <- content(answer, as = 'parsed')

for (i in seq_along(x)){
  if ( is.null(x[[i]]$lastName) == FALSE && x[[i]]$lastName == lastName
) {
    print(x[[i]]$email)
  }
}

## CASE: non-unique last name or no contact

## variables
lastName <- 'Nicolaus'
firstName <- 'Rita'

## calling api
answer <- GET(link)
x <- content(answer, as = 'parsed')

## loop over entire list
for (i in seq_along(x)){
  if ( is.null(x[[i]]$lastName) == FALSE && x[[i]]$lastName == lastName
&& x[[i]]$firstName == firstName ) {
    print(x[[i]]$email)
  }
}

## + ===== | xxx |
## ===== + ##
## '''
## USE CASE 1: We are looking for a list of items that have a certain user
in their contact list.
## 1.2. We have info from 1.1 but need that user only with a certain
contact role (e.g. 'Principal Investigator').
## '''
## + ===== | xxx |
## ===== + ##

## which user should be searched?
## remember 'email' from 1.1?
#email = 'sandra.tippenhauer@awi.de'

## what role should be filtered?
inputRole <- 'Principal Investigator'
inputRole <- 'Editor'

lstID <- NULL
lstURN <- NULL

for (item in seq_along(itemList)){
  link <- paste0(sensorURL, "contacts/getDeviceContacts/", itemList
[[item]])
  print(link)
  answer <- GET(link)
  x <- content(answer, as = 'parsed')

  ## create dictionaries to map uuids to vocables
  dictRoles <- NULL
  dictAccounts <- NULL
  for (i in seq_along(x)){
    ## print(x[[i]])

```

```

        if (is.list(x[[i]]$contact) == TRUE){
            dictAccounts[x[[i]]$contact$'@uuid'] <- x[[i]]$contact$email
        }
        if (is.list(x[[i]]$vocable) == TRUE){
            dictRoles[x[[i]]$vocable$'@uuid'] <- x[[i]]$vocable$generalName
        }
    }
    ## extracting from item
    user <- NULL
    role <- NULL
    for (i in seq_along(x)){
        if (is.list(x[[i]]$vocable) == TRUE){
            role <- append(role, x[[i]]$vocable$'@uuid')
        } else {
            role <- append(role, x[[i]]$vocable)
        }
        ##
        if (is.list(x[[i]]$contact) == TRUE){
            user <- append(user, x[[i]]$contact$'@uuid')
        } else {
            user <- append(user, x[[i]]$contact)
        }
    }
    ## merging info from item and mapping dict
    for (i in seq_along(x)){
        if ( dictRoles[role[i]] == inputRole && dictAccounts[user[i]] ==
email){
            lstID <- append(lstID, item)
            lstURN <- append(lstURN, x[[1]]$item$urn)
        }
    }
}

## neat output as some sort of report ...
print(
    toJSON(list(user = email,
                timeOfQuery = now(tz = "GMT"),
                role = inputRole,
                urn = lstURN,
                id = lstID
            )
        )
)

## ... or just as a single list if item IDs
print(lstID)

## eof

```

Use Case 2: write a certain event (e.g. info about calibration) to item xy

- [] PY
- [] R

input:

- item URN
- user credentials
- event infos

1. get /sensors/item /getItemByUrn/{urn} --> filter for "id"

use case 2 Python

use case 2 R

2. create a usable token post
[input] /sensors
/contacts/login (input is a
json holding username and
password) --> response -->
save token
3. get /sensors/events
/getAllEventTypes -->
filter for event type id for input
in #5.5
4. create json for other input for
#5
5. put /sensors/events
/putEvent/{deviceId} +
INPUT {
 - a. "startDate": "2020-01-01T14:10:00",
 - b. "endDate": "2020-01-01T14:10:00",
 - c. "label": "Test",
 - d. "description": "string",
 - e. "eventType": 0,
 - f. "longitude": 0,
 - g. "latitude": 0,
 - h. "elevationInMeter": 0,
 - i. all children <- not part of the json body
 - j. create version <- not part of the json body [TRUE/FALSE]

}

Use Case 3: create a list of all items of a platform that measure parameter xyz

- [x] PY
- [x] R

Precondition: only valid for current configuration. If the item was object to modifications (mounts) the results may differ.

input:

- top most item
- paramter type (e.g. salinity)

1. get https://sensor.awi.de/rest/sensors/item/{itemID}/getChildrenOfItem/{itemID}
2. get https://sensor.awi.de/rest/sensors/sensorOutputs/getAllSensorOutputTypes
-> filter by i.e. regex "temp"
(water temperature, technical temperature, etc. ...)
3. get https://sensor.awi.de/rest/sensors/sensorOutputs/getDeviceSensorOutputs/{itemID}

TODO: solve outputType is matching exactly but the response is not unique.... *doh!*

use case 3 Python

```
'''
basic documentation of sensor.awi.de REST-API can be found here:
https://sensor.awi.de/api/
'''

## necessary libraries
import json
import requests
import re
import itertools

## PROD
#sensorURL = "https://sensor.awi.de/rest/sensors/"
## SANDBOX
sensorURL = "https://sandbox.sensor.awi.de/rest/sensors/"

## + ===== | xxx |
+ ##
'''
USE CASE 3: give all items that measure parameter xyz on platform zyx
'''

## + ===== | xxx |
+ ##

## ----- 1 ----- ##

urn = 'vessel:polarstern'
#urn = 'vessel:uthoern'
#urn = 'aircraft:polar7'
link = sensorURL + '/item/getItemByUrn/' + urn
answer = requests.get(link)
## check
answer.status_code
item = json.loads(answer.content)

## aux function, saves code later
def getKids(ID):
    link = sensorURL + 'item/getChildrenOfItem/' + str(ID)
    answer = requests.get(link)
```

```

x = json.loads(answer.content)
y = [f['id'] for f in x]
return(y)

## there might be smarter ways ...
idCollection = [item['id']]
initCount = 0
afterCount = len(idCollection)
while initCount != afterCount:
    ## print(idCollection)
    ## retrieve kids
    a = [getKids(ii) for ii in idCollection]
    ## flatten the list and remove '[]'
    subs = list(itertools.chain(*a))
    ## set first counter
    initCount = len(idCollection)
    ## combine lists
    idCollection = idCollection + subs
    ## kickout duplicates
    idCollection = list(dict.fromkeys(idCollection))
    ## set second counter
    afterCount = len(idCollection)

## ----- 2 ----- ##

## should be unique
## see 'thermal diffusivity' vs. 'thermosalinograph'
pattern = 'temperature'

link = sensorURL + "vocabulary/getAllVocables?pointInTime=2015-01-01T00%
3A00%3A00.000Z"
answer = requests.get(link)
x = json.loads(answer.content)

param = []
for i in x:
    ## first occurrence gives uuid
    if isinstance(i['vocableGroup'], dict) and i['vocableGroup']['name']
== 'SensorOutputTypes':
        vocGroup = i['vocableGroup']['@uuid']
        ## increase chance for success by case insensitive matching
        if str(pattern).lower() in str(i['systemName']).lower():
            ## guarantee 1st occurrence in list
            if isinstance(i['vocableGroup'], dict) and i['vocableGroup']
['@uuid'] == vocGroup:
                param.append(i)
            ## guarantee all other occurrences in list
            if isinstance(i['vocableGroup'], str) and i['vocableGroup'] ==
vocGroup:
                param.append(i)

## if not unique ( len(param) == 1) stop here
if len(param) != 1:
    print('result must be unique (as in a single result)')
    print('')
    raise Exception('break')
else:
    print(param)

## ----- 3 ----- ##
param = [param[0]]

parameterID = []
## remember me?
for i in idCollection:
    print(i)
    link = sensorURL + "sensorOutputs/getDeviceSensorOutputs/" + str(i)
    answer = requests.get(link)
    x = json.loads(answer.content)
    ## creating dict to map values

```

```

dic = {}
## just to create dict
for output in x:
    if isinstance(output['sensorOutputType'], dict):
        dic[output['sensorOutputType']['systemName']] = output
['sensorOutputType']['@uuid']
## checking if par exists
if param[0]['systemName'] not in dic:
    pass
else:
    ## true run
    for output in x:
        ## 1st ofccurence
        if isinstance(output['sensorOutputType'], dict) and output
['sensorOutputType']['@uuid'] == dic[param[0]['systemName']]:
            parameterID.append(output['id'])
        ## all other occurences
        elif isinstance(output['sensorOutputType'], str) and output
['sensorOutputType'] == dic[param[0]['systemName']]:
            parameterID.append(output['id'])
        ## something weird going on
        else:
            pass

if len(parameterID) != 0:
    print('found something')

## optional:
## ----- 4 ----- ##

for jj in parameterID:
    link = sensorURL + 'sensorOutputs/getSensorOutput/' + str(jj)
    answer = requests.get(link)
    x = json.loads(answer.content)
    print(x)

## eof

```

use case 3 R

```

## '''
## basic documentation of sensor.awi.de REST-API can be found here:
https://sensor.awi.de/api/
## '''
settingUpThings <- function(x){
    mirror <- 'https://ftp.gwdg.de/pub/misc/cran'
    newPackages <- x[!(x %in% installed.packages()), 'Package']]
    if(length(newPackages)){
        install.packages(newPackages, dep = TRUE, repos = mirror)
    }
    lapply(x, library, character.only = TRUE, logical.return = TRUE)
}

libsToBeLoaded <- c('jsonlite', 'plyr', 'stringr', 'httr')
settingUpThings(libsToBeLoaded)

## PROD
#sensorURL<- "https://sensor.awi.de/rest/sensors/"
## SANDBOX
sensorURL <- "https://sandbox.sensor.awi.de/rest/sensors/"

## + ===== | xxx |
===== + ##
## '''
## USE CASE 3: give all items that measure parameter xyz on platform zyx
## '''
## + ===== | xxx |

```

```

===== + ##

## ----- 1 ----- ##

urn <- 'vessel:polarstern'
#urn <- 'vessel:uthoern'
#urn <- 'aircraft:polar7'
link <- paste0(sensorURL, '/item/getItemByUrn/', urn)
answer <- GET(link)
## check
answer$status_code

item <- content(answer, as = 'parsed')

getKids <- function(ID){
  link <- paste0(sensorURL, 'item/getChildrenOfItem/', ID)
  answer <- GET(link)
  x <- content(answer, as = 'parsed')
  y <- NULL
  for (f in seq_along(x)){
    y[f] <- x[[f]]$id
  }
  return(y)
}

## there might be smarter ways ...
idCollection <- c(item$id)
initCount <- 0
afterCount <- length(idCollection)
while (initCount != afterCount){
  print(idCollection)
  ## set first counter
  initCount <- length(idCollection)
  ## retrieve kids and combine lists
  for (ii in seq_along(idCollection)){
    idCollection <- c(idCollection, getKids(idCollection[ii]))
  }
  ## skipping some foo, not necessary here
  ## ...
  ## kickout duplicates
  idCollection <- unique(idCollection)
  ## set 2nd counter
  afterCount <- length(idCollection)
}

## ----- 2 ----- ##

## should be unique
## see 'thermal diffusivity' vs. 'thermosalinograph'
pattern = 'tempe'

link <- paste0(sensorURL, "vocabulary/getAllVocables?pointInTime=2015-01-
01T00%3A00%3A00.000Z")
answer <- GET(link)
x <- content(answer, as = 'parsed')

param <- list()
for (i in seq_along(x)){
  ## 1st occurrence gives uuid
  if ( is.list(x[[i]]$vocableGroup) == TRUE && x[[i]]$vocableGroup$name
== 'SensorOutputTypes' ){
    vocGroup <- x[[i]]$vocableGroup$@uuid'
  }
  ## increase chance for success by case insensitive matching
  if ( str_detect(tolower(x[[i]]$systemName), tolower(pattern)) ){
    ## guarantee 1st occurrence in list
    if ( is.list(x[[i]]$vocableGroup) == TRUE && x
[[i]]$vocableGroup$@uuid' == vocGroup ){
      param[[length(param)+1]] <- x[[i]]
    }
    ## guarantee all other occurrences in list

```

```

        if( is.character(x[[i]]$vocableGroup) == TRUE && x
[[i]]$vocableGroup == vocGroup ){
            param[[length(param)+1]] <- x[[i]]
        }
    }
}

if ( length(param) != 1 ){
    print('result must be unique (as in a single result)')
} else {
    print(param)
}

## ----- 3 ----- ##

param <- param[[1]] ## only temp, only for now

parameterID <- NULL
## remember me?
for (i in seq_along(idCollection)){
    link <- paste0(sensorURL, "sensorOutputs/getDeviceSensorOutputs/", 61)
# idCollection[i])
    link <- paste0(sensorURL, "sensorOutputs/getDeviceSensorOutputs/",
idCollection[i])
    answer <- GET(link)
    x <- content(answer, as = 'parsed')
    ## creating dict to map values
    dic <- list()
    ## just to create dict
    for (output in seq_along(x)){
        ## print(i)
        if ( is.list(x[[output]]$sensorOutputType) == TRUE ){
            dic[x[[output]]$sensorOutputType$systemName] <- x
[[output]]$sensorOutputType$'@uuid'
        }
    }
    ## checking if par exists
    if ( param$systemName %in% names(dic) == TRUE ){
        ## true run
        for (output in seq_along(x)){
            ## 1st occurrence
            if ( is.list(x[[output]]$sensorOutputType) == TRUE && x
[[output]]$sensorOutputType$'@uuid' == dic[param$systemName] ){
                parameterID <- append(parameterID, x[output][[1]]$id)
            }
            ## all other occurrences
            else if ( is.character(x[[output]]$sensorOutputType) == TRUE
&& x[[output]]$sensorOutputType == dic[param$systemName] ){
                parameterID <- append(parameterID, x[output][1]$id)
            } else {
                next
            }
        }
    }
}

if ( length(parameterID) > 0 ){
    print('found something')
}

## optional:
## ----- 4 ----- ##

for (jj in seq_along(parameterID)){
    link <- paste0(sensorURL, 'sensorOutputs/getSensorOutput/', parameterID
[jj])
    answer <- GET(link)
    x <- content(answer, as = 'parsed')
    print(x)
}

```

```
## eof
```

Use Case 4: adding/removing/modifying a parameter to item xyz

- [X] PY
- [] R

input:

- parental item ID
 - some user added info, such as name and comment
 - parameter id (derived from dict)
 - unit id (derived from dict)
 - (own) user credentials
1. token
 2. get `https://sandbox.sensor.awi.de/rest/sensors/sensorOutputs/getAllSensorOutputTypes` -> filter
 3. get `https://sandbox.sensor.awi.de/rest/sensors/unitsOfMeasurement/getAllUnitsOfMeasurement` -> filter
 4. (if necessary) get `https://sandbox.sensor.awi.de/rest/sensors/item/getItemByUrn/{URN}` -> derive ID

use case 4 Python

```
'''
basic documentation of sensor.awi.de REST-API can be found here:
https://sensor.awi.de/api/
'''

## necessary libraries
import json
import requests

## PROD
#sensorURL = "https://sensor.awi.de/rest/sensors/"
## SANDBOX
sensorURL = "https://sandbox.sensor.awi.de/rest/sensors/"

## + ===== | xxx |
===== + ##
'''
USE CASE 4: adding/removing/modifying a parameter to item xyz
'''
## + ===== | xxx |
===== + ##

## ----- preparation
----- ##

## creating a token, to be used to all 3 operations
userName = 'norbert.anselm@awi.de'
PW = '' # <- sensitive information, do not put in git, spaces or
elsewhere!!!

auth = requests.post(sensorURL + 'contacts/login',
                      data = {'username': userName, 'authPassword': PW}
)

## check if the request was successful
if auth.status_code == 200:
    print('we are happy')
    ## extracting token -> keep it
    token = auth.cookies['x-auth-token']
else:
    print('nope, no token available')

## find out id -- skip if you already know
urn = 'anemometer:aaaa'
link = sensorURL + '/item/getItemByUrn/' + urn
answer = requests.get(link)

## check
answer.status_code

a = json.loads(answer.content)
itemID = a['id']

## ----- adding
----- ##

## this is my input variable
myOutputType = 'grav'

## find proper output type
link = sensorURL + 'sensorOutputs/getAllSensorOutputTypes'
answer = requests.get(link)
```

```

outputs = json.loads(answer.content)

## write to dict
dictOutputTypes = {}
for i in outputs:
    dictOutputTypes[i['generalName']] = [i['id'], i['description']]

## 1. exact match, literally 'in' == 'out'
dictOutputTypes[myOutputType]

## 2. fuzzy match, > 1 hit possible
for i in dictOutputTypes:
    if str(myOutputType).lower() in i.lower():
        #print(i)
        print('id: ' + str(dictOutputTypes[i][0]) + '\t-> ' + i)

## okay, now I know I need 67 (water temperature)
myOutputTypeID = 500

## this is my other input variable
myUnit = 'Celsius'
## now the units
link = sensorURL + 'unitsOfMeasurement/getAllUnitsOfMeasurement'
answer = requests.get(link)
units = json.loads(answer.content)

## 1st approach -- map on longname
dictUnits = {}
for i in units:
    dictUnits[i['longName']] = i['id']

for i in dictUnits:
    if str(myUnit).lower() in i.lower():
        print('id: ' + str(dictUnits[i]) + '\t-> ' + i)

## this is my other input variable -- again ;)
myUnit = 'Cel'
## 2nd approach -- map on ucum, if you know...
dictUnits = {}
for i in units:
    dictUnits[i['ucum']] = i['id']

## mask i with str, otherwise 'None' breaks everythin
for i in dictUnits:
    if str(myUnit) in str(i):
        print('id: ' + str(dictUnits[i]) + ' -> ' + i)

'''
## either way, by longname and by ucum code the unit ID for 'degree
Celsius' is 14, qed
'''

## okay, it is 14
myUnitID = 49

## combine all info on the new parameter
itemParameter = json.dumps({
    "name": "water temperature of a SBE38",
    ##"name": "another var",
    "type_vocableID": myOutputTypeID,
    "unitOfMeasureID": myUnitID,
    "shortName": "temperature_sbe38",
    ##"shortName": "another_var",
    "comment": "no coffee, no jokes"
})

## put all together
s = requests.put(sensorURL + 'sensorOutputs/putSensorOutput/' + str(itemID)
                , data = itemParameter
                , headers = {"content-type": "application/json"}
                , cookies = {'x-auth-token': token})

```

```

)

## check
s.status_code

## ----- modifying
----- ##

'''
in order to modify an item parameter the entire "itemParameter" must be
submitted again,
hence it is easier to retrieve the current state and modify according to
ones needs
'''

## itemID 10877, see above, if output ID is known, skip this part
link = sensorURL + 'sensorOutputs/getDeviceSensorOutputs/' + str(itemID)
answer = requests.get(link)
outputs = json.loads(answer.content)

## pick out the desired one
for i in outputs:
    print(str(i['id']) + '\t' + i['shortname'])
    ## this one works only with unique output types and units
    # print(str(i['id']) + '\t' + i['shortname'] + '\t' + i
['sensorOutputType']['generalName'] + '\t' + i['unitOfMeasurement']
['longName'])

outputID = 93335

## retrieve full info of parameter from list
a = [i for i in outputs if i['id'] == outputID]

## should be single item list
if len(a) == 1:
    b = a[0]
else:
    raise Exception('empty or >1')

## create the api call
link = sensorURL + 'sensorOutputs/modifySensorOutput/' + str(outputID) + '
/' + str(itemID)

## technically this parameter remains unchanged in this case, except for
the comment
itemParameter = json.dumps({
    "name": b['name'],
    "type_vocableID": b['sensorOutputType']['id'],
    "unitOfMeasureID": b['unitOfMeasurement']['id'],
    "shortName": b['shortname'], ## cannot be modified but must be
submitted
    "comment": "but I can have black tea as a substitution"
})

## put == modify the parameter
a = requests.put(link,
                headers = {'content-type': 'application/json'},
                data = itemParameter,
                cookies = {'x-auth-token': token}
)

## check
a.status_code

## ----- removing
----- ##

## recycled IDs from above
link = sensorURL + 'sensorOutputs/deleteSensorOutputFromDevice/' + str
(outputID) + '/' + str(itemID)

```

```

## call api
a = requests.delete(link, headers = {'content-type': 'application/json'},
cookies = {'x-auth-token': token})

## check
a.status_code

## EOF

```

use case 4 R

Use Case 5: create a list for my item that holds necessary infos, such as 'parameter name', 'parameter type', 'unit' and comments', for a PANGAEA data publication.

- [X] PY
- [X] R

input:

- item id
1. getting all outputs from certain item: /sensors
/sensorOutputs
/getDeviceSensorOutputs
/{deviceId} --> scrape all IDs to a list
 2. iterate over list from 1. and obtain each parameter via /sensors/sensorOutputs
/getSensorOutput/{parameterID} --> collect infos on
 - (short) name of parameter
 - parameter type
 - standardized /controlled vocabulary for parameter type (at best a NERC link)
 - unit
 - unit in [ucum](#) format
 - description of parameter
 - (user) comment on parameter
 3. compile 2. to a dataframe
 4. write to hd

use case 5 Python

```

'''
basic documentation of sensor.awi.de REST-API can be found here:
https://sensor.awi.de/api/
'''

import json
import requests
import datetime
import pandas as pd

## PROD
#sensorURL = "https://sensor.awi.de/rest/sensors/"
## SANDBOX
sensorURL = "https://sandbox.sensor.awi.de/rest/sensors/"

## + ===== | xxx |
## ===== + ##
'''
## USE CASE 5: We would like to prepare a list of relevant information
## about an item's parameters,
## such as parameter name, output type, unit, comment, etc. in order to
## add it to my PANGAEA submission
'''
## + ===== | xxx |
## ===== + ##

## 2nd approach <- more realistic! ;)
## starting from numeric id, otherwise resolve to from URN
myItemID = 3654
answer = requests.get(sensorURL + "/item/getItem/" + str(myItemID))
item = json.loads(answer.content)
## am I right??
#print(item['urn'])

## retrieve all parameters
link = sensorURL + "sensorOutputs/getDeviceSensorOutputs/"
answer = requests.get(link + str(myItemID))
b = json.loads(answer.content)
allPars = [j['id'] for j in b]

pHead = []
for i in allPars:
    link = sensorURL + "sensorOutputs/getSensorOutput/" + str(i)
    answer = requests.get(link)
    b = json.loads(answer.content)
    ## if there is a standardized vocable for type -> use it
    if b['sensorOutput']['sensorOutputType']['vocabularyID'] == 'NERC':
        conVocP = ' output: ' + b['sensorOutput']['sensorOutputType']
    ['vocabularyValue'] + ';'
    else:
        conVocP = ' output: -' + ';'
    ## if there is a standardized vocable for item -> use it

```

```

if item['subItemType']['vocabularyID'] == 'NERC':
    conVocI = ' item: ' + item['subItemType']['vocableValue'] + ';'
else:
    conVocI = ' item: -;'
## if there is a comment -> use it
if b['sensorOutput']['comment'] is None:
    customComment = ' comment: -'
else:
    customComment = ' comment: ' + b['sensorOutput']['comment']
## create header
pHead.append(b['sensorOutput']['shortname'] + ' [' + b['sensorOutput']
['unitOfMeasurement']['code'] + ' ] /*' + conVocP + conVocI +
customComment + '//$ no freaking method id')
##
x = pd.DataFrame({'sensorOutputIDs':allPars, 'pangaeaHead': pHead})

## pangaea can only handle tsv -> do it <= HAHA
print(x.to_csv(sep = '\t'))

## eof

```

use case 5 R

```

## '''
## basic documentation of sensor.awi.de REST-API can be found here:
## https://sensor.awi.de/api/
## '''
settingUpThings <- function(x){
  mirror <- 'https://ftp.gwdg.de/pub/misc/cran'
  newPackages <- x[!(x %in% installed.packages()[, 'Package'])]
  if(length(newPackages)){
    install.packages(newPackages, dep = TRUE, repos = mirror)
  }
  lapply(x, library, character.only = TRUE, logical.return = TRUE)
}

libsToBeLoaded <- c('jsonlite', 'httr')
settingUpThings(libsToBeLoaded)

## PROD
#sensorURL<- "https://sensor.awi.de/rest/sensors/"
## SANDBOX
sensorURL <- "https://sandbox.sensor.awi.de/rest/sensors/"

## + ===== | xxx |
## ===== + ##
## '''
## ## USE CASE 5: We would like to prepare a list of relevant information
## about an item's parameters,
## such as parameter name, output type, unit, comment, etc. in order to
## add it to my PANGAEA submission
## '''
## + ===== | xxx |
## ===== + ##

## 2nd approach <- more realistic! ;)
## starting from numeric id, otherwise resolve to from URN
myItemID <- 3654
answer <- GET(paste0(sensorURL, "/item/getItem/", myItemID))
item <- content(answer, as = 'parsed')
## am I right?
print(item$urn)

## retrieve
link <- paste0(sensorURL, "sensorOutputs/getDeviceSensorOutputs/",
myItemID)

```

```

answer <- GET(link)
b <- content(answer, as = 'parsed')

allPars <- NULL
for (j in seq_along(b)){
  allPars[j] <- b[[j]]$id
}

pHead <- NULL
for (i in seq_along(allPars)){
  link <- paste0(sensorURL, "sensorOutputs/getSensorOutput/", allPars[i])
  answer <- GET(link)
  b <- content(answer, as = 'parsed')
  ## if there is a standardized vocable for type -> use it
  if ( b$sensorOutput$sensorOutputType$vocabularyID == "NERC" ){
    conVocP <- paste0('output: ',
b$sensorOutput$sensorOutputType$vocabularyValue, ';')
  } else {
    conVocP <- paste0('output: - ;')
  }
  ## if there is a standardized vocable for item -> use it
  if ( item$subItemType$vocabularyID == "NERC" ){
    conVocI <- paste0(' item: ', item$subItemType$vocabularyValue, ';')
  } else {
    conVocI <- paste0(' item: - ;')
  }
  ## if there is a comment -> use it
  if ( is.null(b$sensorOutput$comment) == TRUE ){
    customComment <- paste0(' comment: -')
  } else {
    customComment <- paste0(' comment: ',b$sensorOutput$comment)
  }
  ## create header
  pHead <- append(pHead, paste0(b$sensorOutput$shortname, ' [',
b$sensorOutput$unitOfMeasurement$code, ']' /*', conVocP, conVocI,
customComment, '//$ no method id'))
}

print(pHead)

## eof

```

Use Case 6: adding /removing a user with a certain role to an item

- [x] PY
- [x] R

input:

- (own) user credentials
 - contact role of a user to be added/removed
 - id of a user to be added /removed
1. post <https://sensor.awi.de/rest/sensors/contacts/login>
 2. get <https://sensor.awi.de/rest/sensors/contacts/getAllContactRoles>
 3. get <https://sensor.awi.de/rest/sensors/item/getItemByUrn/>
 4. user id:
 - a. direct input

use case 6 Python

```

'''
basic documentation of sensor.awi.de REST-API can be found here:
https://sensor.awi.de/api/
'''

## necessary libraries
import re
import json
import requests

## ----- 1 ----- ##

userName = 'norbert.anselm@awi.de'
PW = # <- sensitive information, do not put in git, spaces or elsewhere!!!

## POST request to sensor
## with payload username and password <- this is plain text, keep it as
secure as possible!!
## PROD
sensorURL = 'https://sensor.awi.de/rest/'
## SANDBOX
sensorURL = 'https://sandbox.sensor.awi.de/rest/'
auth = requests.post(sensorURL + 'sensors/contacts/login',

```

- b. derived from mail
- c. from first and last name

add

1. put https://sensor.awi.de/rest/sensors/contacts/putExistingContactToDevice/{itemID}/{userID}/{contactRoleID}

remove

1. delete https://sensor.awi.de/rest/sensors/contacts/deleteContactFromDevice/{itemID}/{userID}/{contactRoleID}

```

data = {'username': userName, 'authPassword': PW}
)

## check httpstatuses.com/ for translation

## check if the request was successful
if auth.status_code == 200:
    print('we are happy')
    ## extracting token -> keep it
    token = auth.cookies['x-auth-token']
else:
    print('nope, no token available')

## ----- 2 ----- ##

link = sensorURL + 'sensors/contacts/getAllContactRoles'

answer = requests.get(link)
allRoles = json.loads(answer.content)

## creating a dict to translate role string representation to numeric
representation
dictRoles = {}
for i in allRoles:
    dictRoles[i['generalName']] = i['id']

## ----- 3 ----- ##

link = sensorURL + 'sensors/item/getItemByUrn/'
## explicitly on sandbox
urn = 'anemometer:aaaa'
#'vessel:uthoern:moses_moblab:ipu443_awi_01'

answer = requests.get(link + urn)
item = json.loads(answer.content)

if auth.status_code == 200:
    itemID = item['id']
else:
    print('no item available')

## ----- 4 ----- ##

## ===== 1 we have user ID ===== ##
userID = 442

x = json.loads(requests.get(sensorURL + 'sensors/contacts/getContact/' +
str(userID)).content)

## checking ...
x['email']

## ===== 2 we know system email ===== ##
user = 'jaeilers@awi.de'
link = sensorURL + 'sensors/contacts/getContactsByEmail/'

x = json.loads(requests.get(link + user).content)

if len(x) == 1:
    userID = x[0]['id']
    print(x)
elif len(x) == 0:
    print('no user')
else:
    ## just for the case
    print('\n choose 1: \n')
    print(x)

## ===== 3 first and last name ===== ##

```

```

link = sensorURL + 'sensors/contacts/getContactsByName/' #norbert/anselm
firstName = 'norbert'
lastName = 'anselm'

x = json.loads(requests.get(link + firstName + '/' + lastName).content)

if len(x) == 1:
    userID = x[0]['id']
elif len(x) == 0:
    print('no user')
else:
    print('\nchoose 1!\n')
    print(x)

## ----- 5 ----- ##

## ===== A -> add certain users ===== ##

## constructing the post link
## input see above!
link = sensorURL + 'sensors/contacts/putExistingContactToDevice/' + str
(itemID) + '/' + str(userID) + '/' + str(dictRoles['Device Contact'])

## submit the cooky
a = requests.put(link, headers = {'content-type': 'application/json'},
cookies = {'x-auth-token': token})

if a.status_code != 200:
    print('we are NOT happy! status code -> ' + str(a.status_code))
else:
    print('we are happy! status code: ' + str(a.status_code) + ' -> ' + a.
reason)

## ===== B -> remove certain users ===== ##

## analogue to put
link = sensorURL + 'sensors/contacts/deleteContactFromDevice/' + str
(itemID) + '/' + str(userID) + '/' + str(dictRoles['Device Contact'])
## hm, wait for SE... <500>
a = requests.delete(link, headers = {'content-type': 'application/json'},
cookies = {'x-auth-token': token})
if a.status_code != 200:
    print('we are NOT happy! status code -> ' + str(a.status_code))
else:
    print('we are happy! status code: ' + str(a.status_code) + ' -> ' + a.
reason)

```

use case 6 R

```

settingUpThings <- function(x){
  ## x == tuple of libs
  ## installing and loading necessary libs
  mirror <- 'https://ftp.gwdg.de/pub/misc/cran'
  ## designated VM path
  ## lib <- '/home/edvs1/nanselm/R/x86_64-pc-linux-gnu-library'
  newPackages <- x[!(x %in% installed.packages()), 'Package']]
  if(length(newPackages)){
    install.packages(newPackages, dep = TRUE, repos = mirror)#, lib =
lib)
  }
  lapply(x, library, character.only = TRUE, logical.return = TRUE)
}

libsToBeLoaded <- c('jsonlite', 'plyr', 'httr')
settingUpThings(libsToBeLoaded)

```

```

## PROD
sensorURL<- "https://sensor.awi.de/rest/"
## SANDBOX
sensorURL <- "https://sandbox.sensor.awi.de/rest/"

## ----- 1 ----- ##

userName <- 'norbert.anselm@awi.de'
##PW <- ##
link <- paste0(sensorURL, 'sensors/contacts/login')

a <- POST(url = link
          , body = list("username" = userName, "authPassword" = PW)
          , encode = "form"
          )

if ( a$status_code == 200 ) {
  print('we are happy')
  token <- a$cookies$value[2]
} else {
  print('nope, no token oavailable')
}

## ----- 2 ----- ##

link <- paste0(sensorURL, 'sensors/contacts/getAllContactRoles')

## JSON way
answer <- read_json(link)
dictRoles <- list()
for (i in seq_along(answer)){
  dictRoles[[answer[[i]]$generalName]] <- answer[[i]]$id
}

## http way
answer <- GET(link)
allRoles <- content(answer, as = 'parsed')
dictRoles <- list()
for (i in seq_along(allRoles)){
  dictRoles[[allRoles[[i]]$generalName]] <- allRoles[[i]]$id
}

## ----- 3 ----- ##

link <- paste0(sensorURL, 'sensors/item/getItemByUrn/')

## explicitly on sandbox
urn = 'anemometer:aaaa'

answer <- GET(paste0(link, urn))

if ( answer$status_code == 200 ) {
  print('we are happy')
  item <- content(answer, as = 'parsed')
  itemID <- item$id
} else {
  print('nope, no item available')
}

## ----- 4 ----- ##

## ===== 1 we have user ID ===== ##
userID <- 442
link <- paste0(sensorURL, 'sensors/contacts/getContact/', userID)

answer <- GET(paste0(link))
x <- content(answer, as = 'parsed')

## test
x$email

```

```

## ===== 2 we know system email ===== ##
user <- 'jaeilers@awi.de'
link <- paste0(sensorURL, 'sensors/contacts/getContactsByEmail/', user)

answer <- GET(paste0(link))
x <- content(answer, as = 'parsed')

if ( length(x) == 1 ){
  userID <- x[[1]]$id
} else if ( length(x) == 0 ){
  print('no user')
} else { ## <- unlikely
  print('choose 1!')
  print(x)
}

## ===== 3 first and last name ===== ##
firstName <- 'janik'
lastName <- 'anselm'
link <- paste0(sensorURL, 'sensors/contacts/getContactsByName/',
  firstName, '/', lastName)

answer <- GET(paste0(link))
x <- content(answer, as = 'parsed')

if ( length(x) == 1 ){
  userID <- x[[1]]$id
} else if ( length(x) == 0 ){
  print('no user')
} else { ## <- unlikely
  print('choose 1!')
  print(x)
}

## ----- 5 ----- ##

## ===== A -> add certain users ===== ##

## constructing the post link
## input see above!
link <- paste0(sensorURL, 'sensors/contacts/putExistingContactToDevice/',
  itemID, '/', userID, '/', dictRoles$'Owner')

a <- PUT(link, body = '{"x-auth-token":token}')

if ( a$status_code != 200 ){
  print(paste0('we are NOT happy! status code -> ', a$status_code))
} else {
  print(paste0('we are happy! status code: ', a$status_code, ' -> '))
}

## ===== B -> remove certain users ===== ##

## analogue to put
link <- paste0(sensorURL, 'sensors/contacts/deleteContactFromDevice/',
  itemID, '/', userID, '/', dictRoles$'Owner')

a <- DELETE(link, body = '{"x-auth-token":token}')
## hm, wait for SE... <500>
if ( a$status_code != 200 ){
  print(paste0('we are NOT happy! status code -> ', a$status_code))
} else {
  print(paste0('we are happy! status code: ', a$status_code, ' -> '))
}

## eof

```

Use Case 7: show all items (e.g. ctd) that measure a certain parameter type

- [] R
- [] PY

input:

Use Case 8: Which item types are available and how are they described

This procedure applies also to sensors/events/getAllEventTypes, sensors/sensorOutputs/getAllSensorOutputTypes, sensors/unitsOfMeasurement/getAllUnitsOfMeasurement, sensors/measurementProperties/getAllMeasurementPropertyTypes, and several more.

- [] R
- [x] PY

input:

- nothing required
1. get `https://sensor.awi.de/rest/sensors/item/getAllItemCategories`
 2. combine all necessary information

use case 7 Python

use case 7 R

use case 8 Python

```
import re
import json
import requests
import pandas as pd

# Use Case 8: Which item types are available and how are they described

## ----- 1 ----- ##

## PROD
sensorURL = 'https://sensor.awi.de/rest/'
## SANDBOX
sensorURL = 'https://sandbox.sensor.awi.de/rest/'

## check httpstatuses.com/ for translation

## retrieve data
link = sensorURL + 'sensors/item/getAllItemCategories'
a = requests.get(link)
b = json.loads(a.content)

## combination of (for you) relevant infos
ids = pd.Series([i['id'] for i in b], name = 'id')
desc = pd.Series([i['description'] for i in b], name = 'description')
name = pd.Series([i['generalName'] for i in b], name = 'generalName')
vocable = pd.Series([i['vocableValue'] for i in b], name = 'vocableValue')

df = pd.concat([ids, name, vocable, desc], axis = 1)

#df.to_json()
#print(df.to_csv( sep = '\t'))

## eof
```

use case 8 R

Use Case 9: explore relevant, human-readable metadata fields of an item

- [] R
- [x] PY

input:

1. get `https://sensor.awi.de/rest/sensors/item/getDetailedItem/{itemID}?includeChildren=true`

use case 9 Python

```
import re
import json
import requests
import pandas as pd

## + ===== | xxx |
## + ===== + ##
'''
USE CASE 9: explore relevant, human-readable metadata fields of an item
'''
## + ===== | xxx |
## + ===== + ##
```

compile parameter list

approach 1

1. map entries, such as output type, unit, etc. -> create lists
2. iterate and align information

approach 2

1. create id list of output items
2. get `https://sensor.awi.de/rest/sensors/sensorOutputs/getSensorOutput/{sensorOutputID}` -> create lists
3. export

make a list of all subitems

1. extract information of choice

For a complete list of all subitems (of all subitems of all subitems), make use of a while loop as in use case no. 3.

As alternative approach the endpoint `https://sensor.awi.de/rest/sensors/item/getChildrenOfItem/{itemID}` could be facilitated as well.

```
## ----- preparation
## -----

## PROD
sensorURL = 'https://sensor.awi.de/rest/'
## SANDBOX
sensorURL = 'https://sandbox.sensor.awi.de/rest/'

itemID = 456 # 4044
## retrieve data (including all subitems)
link = sensorURL + 'sensors/item/getDetailedItem/' + str(itemID) + '?
includeChildren=true'
a = requests.get(link)
b = json.loads(a.content)

## ----- quick exploration
## -----

## which fields are available
b.keys()

## who is the parental item (if available)
b['parentItemLongName']

## the contact element
b['contactRoleItem']

## the associated resources
b['onlineResourceRoleItem']

## how many events ??
len(b['eventItem'])

## ----- compile parameter list
## -----

## -----
## approach 1 -- from json alone
## -----

## create mapping tables for units and output types
dictOutputTypes = {}
dictUnits = {}
for i in b['sensorOutput_Item']:
    if isinstance(i['sensorOutput']['sensorOutputType'], dict):
        ## print(i['sensorOutput']['sensorOutputType']
        ['generalName'])
        dictOutputTypes[i['sensorOutput']['sensorOutputType']['@uuid']] = i
        ['sensorOutput']['sensorOutputType']['vocableValue']
        if isinstance(i['sensorOutput']['unitOfMeasurement'], dict):
            dictUnits[i['sensorOutput']['unitOfMeasurement']['@uuid']] = i
            ['sensorOutput']['unitOfMeasurement']['code']

d, e, f, g, h = [], [], [], [], []

for i in b['sensorOutput_Item']:
    if isinstance(i['sensorOutput']['sensorOutputType'], dict):
        f.append(i['sensorOutput']['sensorOutputType']['vocableValue'])
    else:
        f.append(dictOutputTypes[i['sensorOutput']['sensorOutputType']])
    if isinstance(i['sensorOutput']['unitOfMeasurement'], dict):
        e.append(i['sensorOutput']['unitOfMeasurement']['code'])
    else:
        e.append(dictUnits[i['sensorOutput']['unitOfMeasurement']])
    h.append(i['sensorOutput']['id'])
    g.append(i['sensorOutput']['comment'])
    d.append(i['sensorOutput']['shortname'])
```

```

## as pandas dataframe
pd.DataFrame({'id': h
              , 'shortname': d
              , 'output': f
              , 'unit': e
              , 'comment': g
            })

## -----
## approach 2 -- with 'external' help
## -----

ids = [i['sensorOutput']['id'] for i in b['sensorOutput_Item']]

d, e, f, g, h = [], [], [], [], []

for id in ids:
    link = sensorURL + 'sensors/sensorOutputs/getSensorOutput/' + str(id)
    a = requests.get(link)
    b = json.loads(a.content)
    h.append(id)
    d.append(b['sensorOutput']['shortname'])
    e.append(b['sensorOutput']['unitOfMeasurement']['code'])
    f.append(b['sensorOutput']['sensorOutputType']['vocableValue'])
    g.append(b['sensorOutput']['comment'])

## as pandas dataframe
pd.DataFrame({'id': h
              , 'shortname': d
              , 'output': f
              , 'unit': e
              , 'comment': g
            })

## as json
#x = json.dumps({
#    "id": h,
#    "shortname": d,
#    "output": f,
#    "unit": e,
#    "comment": g
#})

## ----- make a list of all subitems
## -----

## how many (direct) subitems
len(b['childItem'])

ids = [i['id'] for i in b['childItem']]
shortnames = [i['shortName'] for i in b['childItem']]
desc = [i['description'] for i in b['childItem']]

pd.DataFrame({
    'id': ids,
    'shortname': shortnames,
    'description': desc
})

## alternative approach using a different endpoint
link = sensorURL + 'sensors/item/getChildrenOfItem/' + str(itemID)
l = json.loads(requests.get(link).content)

ids = [i['id'] for i in l]
shortnames = [i['shortName'] for i in l]
desc = [i['description'] for i in l]

pd.DataFrame({
    'id': ids,
    'shortname': shortnames,
    'description': desc
})

```

```

})

## eof

```

use case 9 R

Use Case 10: create a new item

- [] R
- [X] PY

input:

- (own) user credentials
 - item type
 - status type
 - some infos about the item to be created
1. post <https://sensor.awi.de/rest/sensors/contacts/login> --> generate auth token
 2. get <https://sensor.awi.de/rest/sensors/item/getAllItemCategories> --> item type ID
 3. get <https://sensor.awi.de/rest/sensors/item/getAllItemStatuses> --> status code ID
 4. put <https://sandbox.sensor.awi.de/rest/sensors/item/createItem?parentItemID={parentItemID}> --> create item

use case 10 Python

```

'''
basic documentation of sensor.awi.de REST-API can be found here:
https://sensor.awi.de/api/
'''

## necessary libraries
import json
import requests
import re

## PROD
#sensorURL = "https://sensor.awi.de/rest/sensors/"
## SANDBOX
sensorURL = "https://sandbox.sensor.awi.de/rest/sensors/"

## + ===== | xxx |
## ===== + ##
'''
    USE CASE 10: create a new item
'''
## + ===== | xxx |
## ===== + ##

## ----- preparation
## ----- ##

## creating a token
userName = 'norbert.anselm@awi.de'
PW = '' # <- sensitive information, do not put in git, spaces or
elsewhere!!!

auth = requests.post(sensorURL + 'contacts/login',
                      data = {'username': userName, 'authPassword': PW}
)

## check if the request was successful
if auth.status_code == 200:
    print('we are happy')
    ## extracting token -> keep it
    token = auth.cookies['x-auth-token']
else:
    print('nope, no token available')

## check
#answer.status_code

## ----- item type id
## ----- ##

link = sensorURL + "item/getAllItemCategories"
answer = requests.get(link)
a = json.loads(answer.content)

## filter device types and create lookup dict
dev = {}
for i in a:
    ## first occurrence gives uuid

```

```

        if isinstance(i['vocableGroup'], dict) and i['vocableGroup']['name']
== 'Device Types':
            devUUID = i['vocableGroup']['@uuid']
            dev[i['generalName']] = i['id']
        elif isinstance(i['vocableGroup'], str) and i['vocableGroup'] ==
devUUID:
            dev[i['generalName']] = i['id']

myItemOfChoice = 'data logger'
## exact match!
for i in dev:
    if str.lower(i) == str.lower(myItemOfChoice):
        print(i + '\t' + str(dev[i]))

myItemOfChoice = 'sample'
## fuzzy match
for i in dev:
    if str.lower(myItemOfChoice) in str.lower(i) :
        print(i + ' ' + str(dev[i]))

'''
item type must be unique, thus manual selection of item type might be
necessary!
'''
## ----- status code id
----- ##

link = sensorURL + 'item/getAllItemStatuses'
answer = requests.get(link)
b = json.loads(answer.content)

status = {}
for i in b:
    status[i['name']] = i['id']

## ----- create payload for API call
----- ##

myStatusOfChoice = 'public'

## mandatory fields marked with ***
description = 'Sphinx of black quartz, judge my vow.'
shortName = 'shortname_of_choice' ***
longName = 'long name with blanks and .. longer' ***
serialNumber = '0815'
manufacturer = 'the manufacturers'
parentID = 0
model = 'B'
inventoryNumber = '4711'
itemStatusID = status[myStatusOfChoice] ***
itemTypeID = dev[myItemOfChoice] ***

'''
parentItemID 0 creates a free floating item, that needs to be assigned to
a parental item. Otherwise the parental ID can be submitted, but beware
the current user needs to have editorial permissions on the parent side.
'''

item = json.dumps({
    "description": description,
    "shortName": shortName,
    "longName": longName,
    "serialNumber": serialNumber,
    "manufacturer": manufacturer,
    "parentID": 0, ## see comment above
    "applicationTypeID": 0,
    "model": model,
    "inventoryNumber": inventoryNumber,
    "itemStatusID": itemStatusID, ***

```

```

        "itemTypeID": itemTypeID, #**
        "id": 0 ## 0 because it's a new one
    })

    ## check
    #print(item)

    ## ----- put item
    ----- ##

    s = requests.put(sensorURL + 'item/createItem?parentItemID='+ str(parentID)
                    , data = item
                    , headers = {"content-type": "application/json"}
                    , cookies = {'x-auth-token': token}
    )

    ## check 201
    #s.status_code

    ## more check
    c = json.loads(s.content)
    d = requests.get(c['uri'])
    d = json.loads(d.content)

    print(d)
    ## eof

```

use case 10 R

Use Case 11: reassign items

- [] R
- [x] PY

input:

- (own) user credentials
 - unmount and mount dates
 - target item ID
 - source item ID
1. post `https://sensor.awi.de/rest/sensors/contacts/login` --> generate auth token
 2. put `https://sensor.awi.de/rest/sensors/itemReassignment/assignItemToItem/{itemID}` + --> content json-body

use case 11 Python

```

'''
basic documentation of sensor.awi.de REST-API can be found here:
https://sensor.awi.de/api/
'''

## necessary libraries
import json
import requests

## PROD
#sensorURL = "https://sensor.awi.de/rest/sensors/"
## SANDBOX
sensorURL = "https://sandbox.sensor.awi.de/rest/sensors/"

## + ===== | xxx |
===== + ##
'''
USE CASE 11: reassign items
'''
## + ===== | xxx |
===== + ##

## ----- preparation
----- ##

## creating a token
userName = 'norbert.anselm@awi.de'
PW = '' # <- sensitive information, do not put in git, spaces or
elsewhere!!!

auth = requests.post(sensorURL + 'contacts/login',
                    data = {'username': userName, 'authPassword': PW}
)

## check if the request was successful

```

```

if auth.status_code == 200:
    print('we are happy')
    ## extracting token -> keep it
    token = auth.cookies['x-auth-token']
else:
    print('nope, no token available')

## check
#answer.status_code

## ----- reassignment
----- ##

## I would like to move my newly created thingy ...
itemID = 18507
## ... to my anemometer
targetID = 10877

## point in time when the item was unmounted, deassembled, removed, etc.
unmountDate = '2023-01-01T20:00:00'
## point in time when the item was mounted, moved, put to the new
environment
mountDate = '2023-01-04T12:34:56'

body = json.dumps({"targetItemID": targetID,
                  "itemID": itemID,
                  "unmount": unmountDate,
                  "mount": mountDate
                })

## doing the job
s = requests.put(sensorURL + 'itemReassignment/assignItemToItem/' + str
(itemID)
                , data = body
                , headers = {"content-type": "application/json"}
                , cookies = {'x-auth-token': token}
            )

## more check
s.status_code

## eof

```

use case 11 R