

**LF** NETWORKING

# Virtual LFN Developer & Testing Forum

---

June 22 - 25, 2020

# Anti-Trust Policy Notice

- › Linux Foundation meetings involve participation by industry competitors, and it is the intention of the Linux Foundation to conduct all of its activities in accordance with applicable antitrust and competition laws. It is therefore extremely important that attendees adhere to meeting agendas, and be aware of, and not participate in, any activities that are prohibited under applicable US state, federal or foreign antitrust and competition laws.
- › Examples of types of actions that are prohibited at Linux Foundation meetings and in connection with Linux Foundation activities are described in the Linux Foundation Antitrust Policy available at <http://www.linuxfoundation.org/antitrustpolicy>. If you have questions about these matters, please contact your company counsel, or if you are a member of the Linux Foundation, feel free to contact Andrew Updegrave of the firm of Gesmer Updegrave LLP, which provides legal counsel to the Linux Foundation.

# Writing tests with Robot Framework

Marek Szwałkiewicz

# Introduction

This talk:

- › is for beginners
- › won't teach Robot Framework itself
- › will get more interesting near the end

# Introduction

- › Grouping and consistency
- › Reusable abstraction
- › Separation of values
- › Setup and teardown
- › `python onap-sdk wrapper`

# Grouping and consistency

Why is it important?

- › It's easier to **find** things in your code
- › It's easier to **understand**
- › so it is easier to **maintain**

*You will see this word a lot in this deck*

Examples:

# Grouping and consistency

```
[user.robot]
```

```
*** Test Cases ***
```

```
Change User Role
```

```
.... Open browser ${URL_PROFILE_PAGE}  
.... Set role ${ROLE_ADMIN}  
.... Submit
```

```
Valid login
```

```
.... Open browser ${URL_LOGIN_PAGE}  
.... Input username Boromir  
.... Input password boro123  
.... Submit
```

```
Valid logout
```

# Grouping and consistency

```
[user.robot]
*** Test Cases ***
Change User Role
... Open browser ${URL_PROFILE_PAGE}
... Set role ${ROLE_ADMIN}
... Submit
Valid login
... Open browser ${URL_LOGIN_PAGE}
... Input username Boromir
... Input password boro123
... Submit
Valid logout
```

```
[role.robot]
```

```
*** Test Cases ***
```

```
Change User Role
```

```
... Open browser ${URL_PROFILE_PAGE}
... Set role ${ROLE_ADMIN}
... Submit
```

```
[user.robot]
```

```
*** Test Cases ***
```

```
Valid login
```

```
... Open browser ${URL_LOGIN_PAGE}
... Input username Boromir
... Input password boro123
... Submit
```

```
Valid logout
```



# Grouping and consistency

```
*** Keywords ***
```

```
Create Service
```

```
....[Documentation] Create a new Service in SDC  
....( ... )  
....[Return] ${service}
```

```
Service Onboarding
```

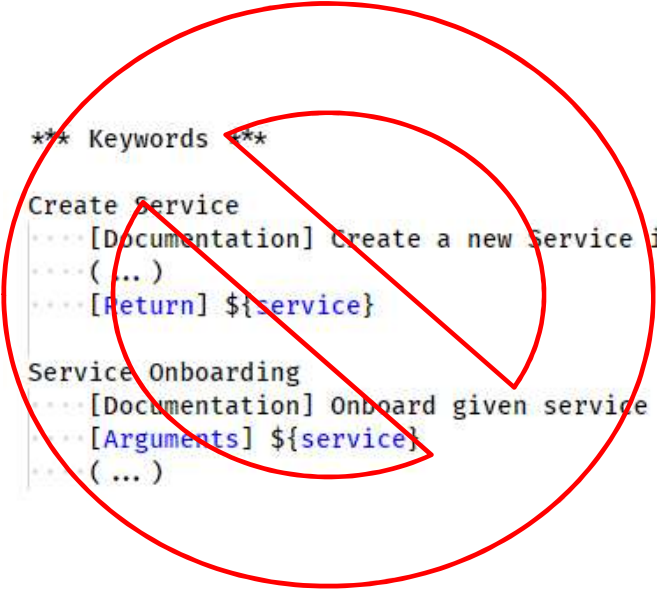
```
....[Documentation] Onboard given service  
....[Arguments] ${service}  
....( ... )
```

# Grouping and consistency

```

*** Keywords ***
Create Service
... [Documentation] Create a new Service in SDC
... ( ... )
... [Return] ${service}

Service Onboarding
... [Documentation] Onboard given service
... [Arguments] ${service}
... ( ... )
    
```



```

*** Keywords ***
Create Service
... [Documentation] Create a new Service in SDC
... ( ... )
... [Return] ${service}

Onboard Service
... [Documentation] Onboard given service
... [Arguments] ${service}
... ( ... )
    
```

# Reusable abstraction

Why is it important?

- › It will make your test **readable**
- › You will create **less code** (sometimes)
- › People with minimal Robot knowledge will be able to **maintain** the tests

Examples:

# Reusable abstraction

```
*** Keywords ***
```

```
Get token
```

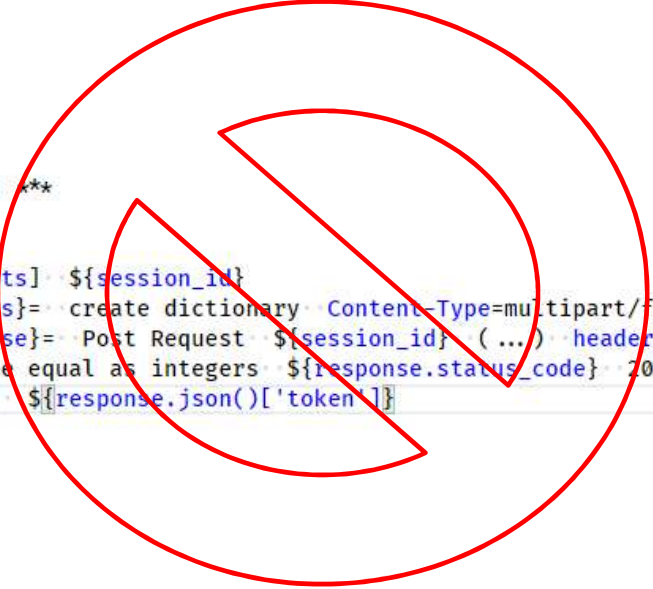
```
... [Arguments] · ${session_id}
... &{headers}= · create dictionary · Content-Type=multipart/form-data
... ${response}= · Post Request · ${session_id} · ( ... ) · headers=${headers}
... should be equal as integers · ${response.status_code} · 200
... [Return] · ${response.json()['token']}
```

# Reusable abstraction

```

*** Keywords ***

Get token
... [Arguments] ${session_id}
... &{headers}= create dictionary Content-Type=multipart/form-data
... ${response}= Post Request ${session_id} ( ... ) headers=${headers}
... should be equal as integers ${response.status_code} 200
... [Return] ${response.json()['token']}
    
```



```

*** Keywords ***
    
```

```

Create form headers
    
```

```

... &{headers}= create dictionary Content-Type=multipart/form-data
... [Return] &{headers}
    
```

```

Get token from response
    
```

```

... [Arguments] ${response}
... [Return] ${response.json()['token']}
    
```

```

Get token
    
```

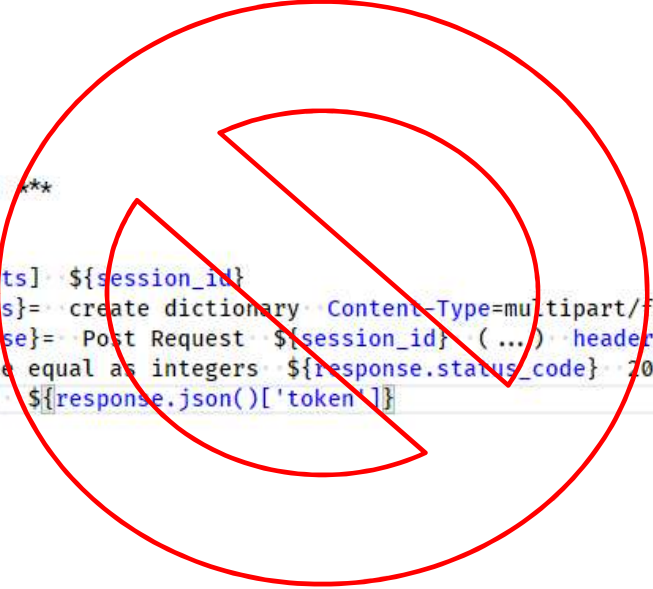
```

... [Arguments] ${session_id}
... &{headers}= Create Form headers
... ${response}= Post Request ${session_id} ( ... ) headers=${headers}
... should be equal as integers ${response.status_code} 200
... [Return] Get token from response ${response}
    
```

# Reusable abstraction

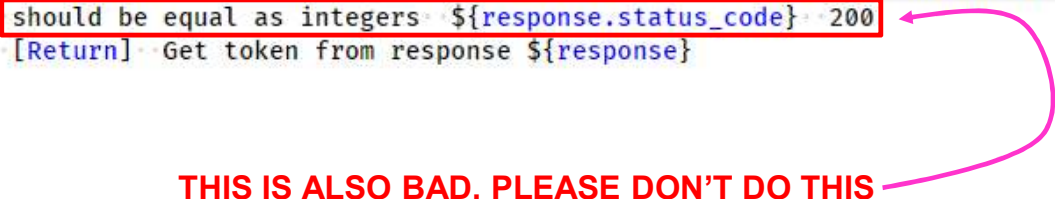
```

*** Keywords ***
Get token
... [Arguments]  ${session_id}
... &{headers}= create dictionary Content-Type=multipart/form-data
... ${response}= Post Request  ${session_id} ( ... ) headers=${headers}
... should be equal as integers  ${response.status_code}  200
... [Return]  ${response.json()['token']}
    
```



```

*** Keywords ***
Create form headers
... &{headers}= create dictionary Content-Type=multipart/form-data
... [Return] &{headers}
Get token from response
... [Arguments]  ${response}
... [Return]  ${response.json()['token']}
Get token
... [Arguments]  ${session_id}
... &{headers}= Create Form headers
... ${response}= Post Request  ${session_id} ( ... ) headers=${headers}
... should be equal as integers  ${response.status_code}  200
... [Return]  Get token from response ${response}
    
```



**THIS IS ALSO BAD, PLEASE DON'T DO THIS**

# Separation of values

Why is it important?

- › Your code will be easier to **customize** and **reuse**
- › It will be much easier to **maintain** the test with up to date data

Examples:

# Separation of values

```
*** Test Cases ***
```

```
Valid login
```

```
... Open browser http://lotr.portal.onap.org/  
... Input username Boromir  
... Input password boro123  
... Submit
```



# Separation of values

```
*** Test Cases ***
```

```
Valid login
```

```
... Open browser http://lotr.portal.onap.org/  
... Input username Boromir  
... Input password boro123  
... Submit
```

```
*** Variables ***
```

```
${URL} ..... http://lotr.portal.onap.org/  
${USERNAME} .. Boromir  
${PASSWORD} .. boro123
```

```
*** Test Cases ***
```

```
Valid login
```

```
... Open browser .. ${URL}  
... Input username .. ${USERNAME}  
... Input password .. ${PASSWORD}  
... Submit
```

# Setup and teardown

Why is it important?

- › You write the boring code once
- › In case of a crash you don't pollute the environment

Examples:

# Setup and teardown

```
*** Test Cases ***
```

```
Certify Service
```

```
....Preload ONAP data
```

```
....( ... )
```

```
Distribute Service
```

```
....Preload ONAP data
```

```
....( ... )
```

# Setup and teardown

```
*** Test Cases ***
```

```
Certify Service  
... Preload ONAP data  
... ( ... )
```

```
Distribute Service  
... Preload ONAP data  
... ( ... )
```

```
*** Settings ***  
Suite Setup ... Preload ONAP data
```

```
*** Test Cases ***
```

```
Certify Service  
... ( ... )
```

```
Distribute Service  
... ( ... )
```

# Setup and teardown

```
*** Test Cases ***
```

```
Certify Service  
... Preload ONAP data  
... ( ... )
```

```
Distribute Service  
... Preload ONAP data  
... ( ... )
```

```
*** Settings ***
```

```
Test Setup ... Preload ONAP data
```

```
Test Teardown ... Remove preloaded data
```

```
*** Test Cases ***
```

```
Certify Service  
... ( ... )
```

```
Distribute Service  
... ( ... )
```

```
Settings ***
```

```
Test Setup ... Preload ONAP data
```

```
Test Cases ***
```

```
Certify Service  
... ( ... )
```

```
Distribute Service  
... ( ... )
```

Bonus Round!

## Python onap-sdk:

```
vendor = Vendor(name=VENDOR_NAME)
vendor.onboard()

vsp = Vsp(name=VSP_NAME, vendor=vendor, package=open(VSP_FILEPATH, "rb"))
vsp.onboard()

vf = Vf(name=VF_NAME, vsp=vsp)
vf.onboard()

service = Service(name=SERVICE_NAME, resources=[vf])
service.onboard()
```

## Robot onap-sdk:

```
${vendor} = Create Vendor ${VENDOR_NAME}
Onboard Vendor ${vendor}

${vsp} = Create Vsp ${VSP_NAME} ${vendor} ${vsp_file_bytes}
Onboard Vsp ${vsp}

${vf} = Create Vf ${VF_NAME} ${vsp}
Onboard Vf ${vf}

@{vfs} = Create List ${vf}
${service} = Create Service ${SERVICE_NAME} @{vfs}
Onboard Service ${service}
```

## Next steps?

- › Publishing of robotframework-onapsdk
- › Extending ONAP wiki pages about Robot with examples
- › Publishing „create your own Robot library” template

[marek.szwalkiewicz@external.t-mobile.pl](mailto:marek.szwalkiewicz@external.t-mobile.pl)



**OLFN**NETWORKING

Virtual Developer & Testing Forum

June 22 - 25, 2020

Thank You!  
Dziękuję!

**OLFN**NETWORKING