McM Scripting

Presented by Justinas Rumševičius

2020

Before we start

- Link to McM: https://cms-pdmv.cern.ch/mcm/
- McM Rest API: https://cms-pdmv.cern.ch/mcm/restapi
- GitHub repo with examples: https://github.com/cms-PdmV/mcm_scripts
- For most actions you will need to have a valid CERN SSO cookie
  - Public APIs do not require a cookie. Public APIs: https://cms-pdmv.cern.ch/mcm/public/restapi/
- You can use either provided python code or do everything with *curl*
- Previous tutorials:
  - https://indico.cern.ch/event/807778/
  - https://indico.cern.ch/event/735384/

McM scripting and cookies

- McM scripting class will try to handle cookies for you
- If no cookie path is specified, McM scripting class will check whether:
  - `~/private/prod-cookie.txt` exists if `dev=False`. If yes, it will use this file as cookie
  - `~/private/dev-cookie.txt` if dev=True. If yes, it will use this file as cookie
- If no cookie path is specified and files mentioned above do not exist, McM scripting class will automatically generate cookie file
- McM scripting class will automatically generate a new cookie up to three times if request to McM fails
- You can also delete cookies, so they could be automatically recreated:
  - `rm ~/private/*-cookie.txt`

Generating CERN SSO cookie

- Use cern-get-sso-cookie command line tool to create cookie file:
  - `cern-get-sso-cookie --url https://cms-pdmv-dev.cern.ch/mcm/ -o dev-cookie.txt`
  - `cern-get-sso-cookie --url https://cms-pdmv.cern.ch/mcm/ -o prod-cookie.txt`
- It is already available in lxplus nodes
- It expires after ~10 hours
- Dev cookie is valid only for dev environment and production cookie is available only for production environment
- More info can be found here:
  - https://linux.web.cern.ch/linux/docs/cernssocookie.shtml

Prerequisites - “Step 0”

- In order to use McM scripting class, you have to import it
- We recommend to use one stored in AFS - always the newest version
- McM scripting class code can be found in github:
  - https://github.com/cms-PdmV/mcm_scripts/blob/master/rest.py
- To use code from AFS, add this to your python script imports:

```
import sys
sys.path.append('/afs/cern.ch/cms/PPD/PdmV/tools/McM/')
from rest import McM
```

- `McM(dev=True)` will use development environment
- `McM(dev=False)` will use production (real-deal) environment
- For more verbosity, you can enable debug printing `McM(debug=True)`

Getting requests

- Using python:

```
mcm = McM(dev=True)
mcm.get('requests', 'PPD-RunIIWinter19PFCalib17pLHE-00001', method='get')
mcm.get('requests', query='prepid=*-RunIIWinter19PFCalib17pLHE-*&status=new')
```

- Using *curl*:

```
curl -s -k --cookie dev-cookie.txt
https://cms-pdmv-dev.cern.ch/mcm/restapi/requests/get/PPD-RunIIWinter19PFCalib17pLHE-00001

curl -s -k --cookie dev-cookie.txt
https://cms-pdmv-dev.cern.ch/mcm/search/?db_name=requests&prepid=*-RunIIWinter19PFCalib17pLHE-*
```

- Four parts: <database name>, <prepid>, <method> (get is default), <query>

Databases in McM

- Databases in McM:
  - batches
  - campaigns
  - chained_campaigns
  - chained_requests
  - flows
  - invalidations
  - lists
  - mccms
  - requests
  - users

Queries in McM

- Queries in McM scripting are the same as you see in your browser URL bar
- Exact value
  - `prepid=PPD-RunIIWinter19PFCalib17GS-00004`
  - `tags=Autumn18P1Moriond19DR`
- Wildcards
  - `prepid=*-RunIIWinter19PFCalib17GS-*`
  - `member_of_chain=PPD-chain_RunIIWinter19PFCalib17GS-*`
- Multiple conditions
  - `pwg=PPD&member_of_campaign=RunIIWinter19PFCalib17GS`
  - `approval=submit&status=submitted`

Getting a request and looking at it

- Using python:

```
import json
import sys
sys.path.append('/afs/cern.ch/cms/PPD/PdmV/tools/McM/')
from rest import McM

# Create McM instance, use default cookie location
mcm = McM(dev=True)
# Prepid of a request
prepid = 'PPD-RunIIWinter19PFCalib17pLHE-00001'
# Get request (dictionary) from McM with "prepid" prepid
req = mcm.get('requests', prepid)
# Print Python dictionary with nice indentation
print(json.dumps(req, indent=4, sort_keys=True))
```

Getting range of requests

- Using python:

```
input_data = """
    B2G-Fall13-00001
    B2G-Fall13-00005 -> B2G-Fall13-00015
"""
mcm.get_range_of_requests(input_data)
```

- Using *curl*:

```
curl -X PUT -s -k --cookie dev-cookie.txt -H "Content-Type: application/json"
https://cms-pdmv-dev.cern.ch/mcm/restapi/requests/listwithfile -d
'{"contents":"B2G-Fall13-00001\nB2G-Fall13-00005 -> B2G-Fall13-00015"}'
```

Note the **\n** in curl's json! Newlines are important here

Creating a new request

- To create a request, two fields are required: PWG and campaign
- Using python:

```
request_dict = {'pwg': 'B2G', 'member_of_campaign': 'PhaseISpring17GS'}
mcm.put('requests', request_dict)
```

- Using *curl*:

```
curl -X PUT -s -k --cookie dev-cookie.txt -H "Content-Type: application/json"
https://cms-pdmv-dev.cern.ch/mcm/restapi/requests/save -d '{"pwg": "B2G",
"member_of_campaign": "PhaseISpring17GS"}'
```

Editing a request

- Using python:

```
request = mcm.get('requests', 'B2G-PhaseISpring17GS-00001')
request['notes'] = 'This is a note'
mcm.update('requests', request)
```

- Using *curl*:

```
curl -X PUT -s -k --cookie dev-cookie.txt -H "Content-Type: application/json"
https://cms-pdmv-dev.cern.ch/mcm/restapi/requests/update -d '{<your modified
request object>}'
```

Cloning a request

- Using python:

```
request = mcm.get('requests', 'B2G-PhaseISpring17GS-00001')
mcm.clone_request(request)
```

- Using *curl*:

```
curl -X PUT -s -k --cookie dev-cookie.txt -H "Content-Type: application/json"
https://cms-pdmv-dev.cern.ch/mcm/restapi/requests/clone -d '{<your modified
request object>}'
```

Creating a new MccM ticket

- Using python:

```
new_mccm = {'pwg': 'B2G', 'prepid': 'B2G', 'requests': [['B2G-Fall13-00005',
'B2G-Fall13-00007'], 'B2G-Fall13-00008']}
mcm.put('mccms', new_mccm)
```

- Using *curl*:

```
curl -X PUT -s -k --cookie dev-cookie.txt -H "Content-Type: application/json"
https://cms-pdmv-dev.cern.ch/mcm/restapi/mccms/save -d '{"prepid":"B2G",
"pwg":"B2G", "requests": ["B2G-Fall13-00008"]}'
```

- Note: list inside list represents a range
- Note: PrepID is same as PWG, but later on it gets overwritten
- Users should check if requests exist before ticket creation

Approving or resetting a request

- Steps: 0 - new, 1 - validation, 2 - define, 3 - approved, 4 - submit, None - approve to next step
- Approve to next step with python:

```
mcm.approve('requests', 'B2G-Fall13-00001', None)
```

- Approve to next step with *curl*:

```
curl -s -k --cookie dev-cookie.txt
https://cms-pdmv-dev.cern.ch/mcm/restapi/requests/approve/B2G-Fall13-00001
```

- Reset with python:

```
mcm.approve('requests', 'B2G-Fall13-00001', 0)
```

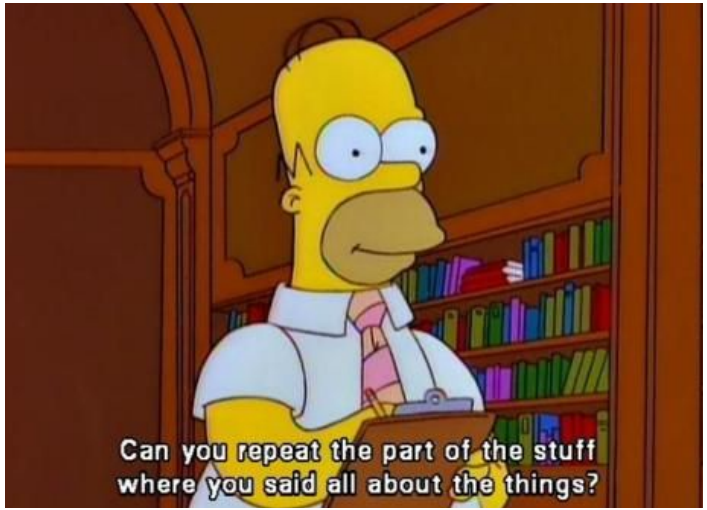
- Reset with *curl*:

```
curl -s -k --cookie dev-cookie.txt
https://cms-pdmv-dev.cern.ch/mcm/restapi/requests/approve/B2G-Fall13-00001/0
```

Tips and tricks: debugging

- Am I in development or production environment? (`mcm = McM(dev=True)`)
- If something should exist, does it really exist? (Check in McM website)
- If something shouldn't exist, does it really not exist? (Check in McM website)
- `print print print`!
- Keep it simple - 10 short lines is better than one long line of code
- Default environment is dev (`McM()` is same as `McM(dev=True)`)
- Play in development, work in production McM
  - Development: `McM(dev=True)` https://cms-pdmv-dev.cern.ch/mcm/
  - Production: `McM(dev=False)` https://cms-pdmv.cern.ch/mcm/

Questions?


Can you repeat the part of the stuff
where you said all about the things?

Exercises!

You thought you will get away without a test…?

Exercise 1

- Getting a request and looking at it:
  - Get this **request** in your script: `PPD-Run3Summer19GS-00001`
  - Print it with `json.dumps(request, indent=4, sort_keys=True)`
- Hint: check slides number 5 and 6

Exercise 2

- Updating a request attribute:
  - Get a random request (don't use `PPD-Run3Summer19GS-00001`)
  - Change notes (string), memory (integer), energy (float) or/and tags (list of strings)
  - Update request in McM with `print(mcm.update('requests', request))`
  - Get the same request again
  - Print it using `json.dumps` and make sure values changed as intended

Exercise 3

- Getting chained requests:
  - Get ALL chained requests that contain `contains=PPD-RunIIAutumn18DR-00016`
  - Print chained request prepids and all requests in these chained requests
  - Expected result:

```
PPD-chain_RunIIFall18GS_..._flowRunIIAutumn18NanoAODv4-00001
  PPD-RunIIFall18GS-00025
  PPD-RunIIAutumn18DR-00016
  PPD-RunIIAutumn18RECOBParking-00002
  PPD-RunIIAutumn18MiniAOD-00004
  PPD-RunIIAutumn18NanoAODv4-00003
PPD-chain_RunIIFall18GS_..._flowRunIIAutumn18NanoAODv5-00001
  PPD-RunIIFall18GS-00025
  PPD-RunIIAutumn18DR-00016
  PPD-RunIIAutumn18RECOBParking-00002
  PPD-RunIIAutumn18MiniAOD-00004
  PPD-RunIIAutumn18NanoAODv5-00001
```

  - Hint: request list in chained request is called "requests"

Exercise 4

- Cloning a request:
  - Get this request in your script: `PPD-Run3Summer19GS-00001`
  - Change PWG attribute to your favourite PWG
  - Clone it using `print(mcm.clone_request(request))`
  - Function in previous step returns a dictionary that contains prepid of new (cloned) request
  - Fetch new request
  - Change memory to something else and notes to `"This is a clone by <your name>"`
  - Update request in McM with `print(mcm.update('requests', request))`
  - Fetch updated request again
  - Print it with `json.dumps(request, indent=4, sort_keys=True)`

Exercise answers

Exercise 1 answer code

```
import sys
import json
sys.path.append('/afs/cern.ch/cms/PPD/PdmV/tools/McM/')
from rest import McM


# McM instance
mcm = McM(dev=True)

request = mcm.get('requests', 'PPD-Run3Summer19GS-00001')
# This works as well:
# request = mcm.get('requests', query='prepid=PPD-Run3Summer19GS-00001')
print(json.dumps(request, indent=4, sort_keys=True))
```

Exercise 2 answer code

```
import json
import sys
sys.path.append('/afs/cern.ch/cms/PPD/PdmV/tools/McM/')
from rest import McM


# McM instance
mcm = McM(dev=True)

request = mcm.get('requests', 'B2G-2019GEMUpg14-00006')
print(json.dumps(request, indent=4, sort_keys=True))
request['notes'] = 'This is a note'
request['energy'] = 9.99
request['memory'] = 15900
request['tags'] = ['Tag1', 'Tag2', 'Tag3']
print(mcm.update('requests', request))
```

Exercise 3 answer code

```
import sys
sys.path.append('/afs/cern.ch/cms/PPD/PdmV/tools/McM/')
from rest import McM


# McM instance
mcm = McM(dev=False)

chained_requests = mcm.get('chained_requests', query='contains=PPD-RunIIAutumn18DR-00016')
for chained_request in chained_requests:
    print(chained_request['prepid'])
    for request in chained_request['chain']:
        print('  %s' % request)
```

Exercise 4 answer code

```
import json
import sys
sys.path.append('/afs/cern.ch/cms/PPD/PdmV/tools/McM/')
from rest import McM


# McM instance
mcm = McM(dev=True)

existing_request = mcm.get('requests', 'PPD-Run3Summer19GS-00001')
existing_request['pwg'] = 'TRK'
clone_result = mcm.clone_request(existing_request)
print(clone_result)
clone_prepid = clone_result['prepid']
cloned_request = mcm.get('requests', clone_prepid)
cloned_request['memory'] = 15900
cloned_request['notes'] = 'This is a clone by Justinas R.'
print(mcm.update('requests', cloned_request))
updated_cloned_request = mcm.get('requests', clone_prepid)
print(json.dumps(updated_cloned_request, indent=4, sort_keys=True))
```