Student Task 5 Dynamic ZABBIX Maps for JumpingJIVE

Final Report

Written by Jingyao Su

Latest version updated on 28.03.2019

Outline:

1. Progress and timeline
2. User Manual
3. Developer Manual

Student Task 5 Dynamic ZABBIX Maps for JumpingJIVE

Progress and timeline

1. Install Ubuntu Linux

Description	Progress	Timeline	Comments
Install Ubuntu Linux on the test computer	100%	09/2018-10/2018	The laptop was picked up in September, 2018, when the installation was started afterwards. Ubuntu 18.04 has already been successfully installed on the test laptop.

2. Getting familiar with Linux commands and operations

Description	Progress	Timeline	Comments
Getting familiar with Linux commands and operations, referring to tutorials and exercises.	100%	08/2018-12/2018	Linux system is the only working platform for this task. Thus operations and commands on Linux are basics for further development.

3. Install Zabbix on Linux

Description	Progress	Timeline	Comments
Install Zabbix on Linux, , mainly based on Zabbix official tutorials and handbooks, as well as JumpingJIVE project documentations including *JumpingJIVE Deliverable 8.5 Integration existing software into central infrastructure (for system monitoring)* and *VLBI SysMon Node*	100%	09/2018-10/2018	Both Zabbix official tutorials and JumpingJIVE project documentations provide detailed procedures for software installation. These two were both refered during the installation process.

4. Setup configuration of Zabbix

Description	Progress	Timeline	Comments
Setup configuration of Zabbix according to project requirements.	100%	09/2018-10/2018	Consecutive process with the next subtask of software configuration.

			Configurations have been set according to the documentations including *JumpingJIVE Deliverable 8.5 Integration existing software into central infrastructure (for system monitoring)* and *VLBI SysMon Nod.* However, as the environment of laptop is not exactly same with server machine, not all steps are deployed with the laptop.

5. Understand logic and structure of Zabbix

Description	Progress	Timeline	Comments
Understand logic and structure of Zabbix. Refer to the tutorials and other documentations to have an overview of Zabbix software. Understand what needs to deal with Zabbix. Understand the performance of Zabbix functions.	100%	10/2018-11/2018	Basic web-based logic has been investigated.

6. Understand operations of Zabbix

Description	Progress	Timeline	Comments
Understand operations of Zabbix. Especially operations related to the target scripts such as map creation, map parameters definition, graph operations as well as checking software information.	100%	10/2018-11/2018	Having learnt and practiced basic operations with GUI. Should further understand more details of operations.

7. Understand the objective of tasks

Description	Progress	Timeline	Comments
Understand the objective of the task by face-to-face meeting.	100%	11/2018	Had a face-to-face meeting with Dr. Neidhart. Some confusions and questions have been answered.

8. Selection of programming language

Description	Progress	Timeline	Comments

Selection of programming language according to current conditions. In addition, get familiar with the language programming if haven't learnt it before.	100%	09/2018-12/2018	Python was chosen because its good performance in terms of efficiency, compatibility and expansibility. Thanks to a basic pre-knowledge about Python language, this step could be finished faster than expected.

9. Understand Zabbix API

Description	Progress	Timeline	Comments
Understand Zabbix API by referring to Zabbix official tutorials.	100%	10/2018-11/2018	Knowledge about Zabbix API could be easily found from official tutorials. This Information is critically fundamental for further developement.

10. Understand JSON-RPC 2.0

Description	Progress	Timeline	Comments
Understand JSON-RPC 2.0	100%	11/2018-12/2018	Having not learnt about JSON-RPC 2.0 before. However, this is also essential because it defines the standard data structures and the rules.

11. Understand usage of JSON-RPC 2.0 in Python

Description	Progress	Timeline	Comments
Understand usage of JSON-RPC 2.0 in Python	100%	11/2018-12/2018	Having understood JSON-RPC in last subtask and learnt how to realize it in python3.

12. Write scripts as requests to get host information from Zabbix

Description	Progress	Timeline	Comments
Write scripts as requests to get information from Zabbix	100%	12/2018-01/2019	Requests through Zabbix API can only be done with login and token information, which should be firstly stated in the scrips. Finished.

13. Write scripts to create a new image

Description	Progress	Timeline	Comments

Write scripts to create a new image	100%	12/2018-01/2019	Stations or antennas should be represented by an icon on the map. Background of the map is realized by an image. In Zabbix API, it is introduced in the form of BASE64. Already finished. Attention further high resolution image is needed.

14. Write scripts to create a new map

Description	Progress	Timeline	Comments
Write scripts to create a new map	100%	12/2018-01/2019	Maps creation can be not only easily performed through GUI but also through python requests with self-defined parameters. Many properties should be defined through python scripts. Finished.

15. Write scripts to add or delete a telescope position

Description	Progress	Timeline	Comments
Write scripts to add or delete a telescope position	100%	12/2018-01/2019	These are basic operations in Zabbix. Finished. Attention that coordinates are just approximated, which needs to be confirmed by project responsible person. In addition, detailed information about telescopes, stations are still unknown. Further adjustment need to be performed.

16. Programming optimization

Description	Progress	Timeline	Comments
Programming optimization. Scripts of single subtasks should be combined and optimized.	75%	02/2019-03/2019	Partly finished. Lack of knowledge about what it is expected to look like. Currently shown as a python script.

17. Adjustment according to productive system VLBI SysMon

Description	Progress	Timeline	Comments
Adjustment according to productive system VLBI SysMon	25%	02/2019-03/2019	Not yet start. Need data and technical support from project responsible person.

18. Test and Integration

Description	Progress	Timeline	Comments
Test and integration on both test laptop and productive system.	50%	02/2019-03/2019	Not yet finished. Testing on laptop is being in process.

Student Task 5 Dynamic ZABBIX Maps for JumpingJIVE User Manual

1. Introduction

1.1 Basics

The scripts are written in Python 3, consisting of 3 .py files:

‘dynamicmap.py’
‘login.py’
‘images.py’

Main functions shall be realized in ‘dynamicmap.py’. The other two work as modules.

1.2 Functions

1. Get host information

2. Create images as background of maps

3. Create Wettzell station map and EVN global map

4. Determine station/antennas location on the map

5. Add or delete stations/antennas on the map

6. Update and delete maps

7. Possibility to switch on/off the antennas on the map

1.3 Methods

1. Python IDE

Scripts can be run with all inputs updated firstly at certain location (Definition part of ‘dynamicmap.py’).

```
# ==============================================================================
# Definitions and login account
# ==============================================================================
# Zabbix server address, only update here
url = 'http://192.168.1.105/zabbix/api_jsonrpc.php' # test Zabbix server on laptop

# Login information, update here if no extra input from command lines
user = "Admin"
password = "zabbix"

# system IDs of images or maps in Zabbix, update here if no extra input from command lines
wettzellimageid = 214
wettzellmapid = 33
evnimageid = 215
evnmapid = 34
hostid = 10084
hostid1 = 0
hostid2 = 0
hostname = "Zabbix server"

# images in BASE64 format
image4000_2000 = images.image4000_2000
image2000_1000 = images.image2000_1000
image1000_500 = images.image1000_500
imagewettzell = images.imagewettzell
```

Fig.1 Definition part of scripts shown in Python IDE

2. Command lines

Scripts can be called and run from Command Prompt (Ubuntu Terminal) by command line argument.

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 6 Admin zabbix 31
Wettzell map deleted successfully
('Request 6 result: ', {u'jsonrpc': u'2.0', u'result': {u'sysmapids': [31]}, u'i
d': 7})
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 9 Admin zabbix 32
EVN global map deleted successfully
('Request 9 result: ', {u'jsonrpc': u'2.0', u'result': {u'sysmapids': [32]}, u'i
d': 10})
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 10 Admin zabbix 32
request not exists . please check again.
```

Fig.2 Command line argument in Ubuntu Terminal

1.4 Environment

Since Python 3 should be already installed in the Ubuntu Linux system, only one third-party Python package (requests) is needed to be installed in extra.

To see which version of Python 3 you have installed, open a command prompt and run

```
$ python3 --version
```

If you are using Ubuntu 16.10 or newer, then you can easily install Python 3.6 with the following commands:

```
$ sudo apt-get update
$ sudo apt-get install python3.6
```

If you’re using another version of Ubuntu (e.g. the latest LTS release), to install Python 3.6:

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt-get update
$ sudo apt-get install python3.6
```

Python 3.4 and later include pip by default. To see if pip is installed, open a command prompt and run

```
$ command -v pip
```

If no pip is installed, run

```
$ apt-get install pip
```

Then to install ‘requests’ package, run

```
$ pip install requests
```

2. Definitions and Zabbix login

Basic parameters should be known and input before execution, including:

‘url’: Zabbix server address
‘user’: User account of Zabbix system
‘password’: User password of Zabbix system
‘wettzellimageid’: background image ID for Wettzell station map
‘wettzellmapid’: map ID of Wettzell station in Zabbix system
‘evnimageid’: background image ID for EVN global map
‘evnmapid’: map ID of EVN global network in Zabbix system
‘hostid’: host ID of telescope/antenna to be monitored
‘hostid1’: host ID of twin telescope 1 at Wettzell station
‘hostid2’: host ID of twin telescope 2 at Wettzell station
‘hostname’: name of host whose information is expected to be obtained

For execution method of Python IDE, all above information should be updated in the ‘Definition and login account’ part of ‘dynamicmap.py’ (Fig.1).

To select which request is expected to run, the sequence number of request should be updated in ‘Main function’ part of ‘dynamicmap.py’ (Fig.3).

```
# ==============================================================================
# Main function
# ==============================================================================
def main():
    global user,password,request_number,hostid1,hostid2,wettzellimageid
    global evnimageid,wettzellmapid,evnmapid
    # login through command line
    if len(sys.argv) >= 4:
        user = sys.argv[2]
        password = sys.argv[3]

    # Selection of request to be deployed
    if len(sys.argv) == 1:
        request_number = 1 # defaut request to be deployed,
                           # update here if no extra input from command lines
    else:
        if sys.argv[1].isdigit():
            request_number = int(sys.argv[1])
        else:
            request_number = 0
```

Fig.3 Main function part of scripts

3. Get host information

Request 1 is used to get host information such as IP address and host ID. Default host is the Zabbix server. Scripts can also be used to get other host information by changing the host name “Zabbix server” to others.

How to use:

1. Python IDE

- Update Zabbix system user account and password, URL of Zabbix server, name of host
- Set request number (line 548) to 1
- Run ‘dynamicmap.py’

2. Command line

Command line argument format:

> “python dynamicmap.py 1 user password hostname”
> Or
> “python dynamicmap.py 1” for default configuration

For example:

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 1 Admin zabbix Zabbix server
('login succeeds. Your host is', u'Zabbix server', 'with ID of ', u'10084', '. I
P address of interfaces is', u'192.168.1.105')
```

Result is the host information including host name, host ID and IP address.

Error analysis:

When host name is wrongly input, the result shows “request error.”

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 1 Admin zabbix 'Zabbix server'
request error. see error message.
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 1 Admin zabbix "Zabbix server"
request error. see error message.
```

When login account is error, the result shows “Not authorised”.

When request number is wrongly input or missed, the result shows “request not exists”

4. Create new images

Request 2 and 3 are used to create images as map background, for Wettzell Station and EVN global network respectively.

How to use:

1. Python IDE

- Update Zabbix system user account and password, url of Zabbix server
- Update image information in BASE64 formation in ‘images.py’

- Set request number (line 548) to 2 or 3
- Run ‘dynamicmap.py’

2. Command line

Command line argument format:

> “python dynamicmap.py 2 user password”
> Or
> “python dynamicmap.py 2” for default configuration

For example:

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 2 Admin zabbix
Wettzell image created successfully
('Request 2 result: ', {u'jsonrpc': u'2.0', u'result': {u'imageids': [u'214']},
u'id': 3})
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 3 Admin zabbix
Global map image created successfully
('Request 3 result: ', {u'jsonrpc': u'2.0', u'result': {u'imageids': [u'215']},
u'id': 4})
```

Result is the image ID of new images created (“wettzellimageid” or “evnimageid”).

Error analysis:

When image formation is wrongly input, the result shows “request error.”

When login account is error, the result shows “Not authorised”.

When request number is wrongly input or missed, the result shows “request not exists”

5. Create new maps

Request 4 and 7 are used to create maps for Wettzell Station and EVN global network respectively.

For Request 4,

How to use:

1. Python IDE

- Update Zabbix system user account and password, url of Zabbix server
- Update image ID of Wettzell station, host ID of twin telescopes at Wettzell station
- Set request number (line 548) to 4
- Run ‘dynamicmap.py’

2. Command line

Command line argument format:

> “python dynamicmap.py 4 user password wettzellimageid hostid1 hostid2”
> Or
> “python dynamicmap.py 4” for default configuration

For example:

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 4 Admin zabbix 214 10084 10084
Wettzell map created successfully
('Request 4 result: ', {u'jsonrpc': u'2.0', u'result': {u'sysmapids': [u'31']},
u'id': 5})
```

Result is the map ID of new created map for Wettzell station(”wettzellmapid”).

Error analysis:

When image ID or host IDs are wrongly input, the result shows “request error”.

When login account is error, the result shows “Not authorised”.

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 4 Admin zabbixx 214 10084 10084
('Request 4 map creates error:', u'Not authorised.')
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 4 Admin zabbix 210 10084 10084
('Request 4 map creates error:', u'SQL statement execution has failed "INSERT IN
TO sysmaps (label_location,height,expandproblem,grid_align,backgroundid,name,lab
el_type_map,width,label_type_host,highlight,userid,sysmapid) VALUES (\'0\',\'300
\',\'0\',\'0\',\'210\',\'Wettzell Station\',\'2\',\'400\',\'0\',\'1\',\'1\',\'33
\')".')
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 4 Admin zabbix 300 10084 10084
('Request 4 map creates error:', u'SQL statement execution has failed "INSERT IN
TO sysmaps (label_location,height,expandproblem,grid_align,backgroundid,name,lab
el_type_map,width,label_type_host,highlight,userid,sysmapid) VALUES (\'0\',\'300
\',\'0\',\'0\',\'300\',\'Wettzell Station\',\'2\',\'400\',\'0\',\'1\',\'1\',\'33
\')".')
```

When request number is wrongly input or missed, the result shows “request not exists”

For Request 7,

How to use:

1. Python IDE

- Update Zabbix system user account and password, url of Zabbix server
- Update image ID of EVN global network, map ID of Wettzell station
- Set request number (line 548) to 7
- Run ‘dynamicmap.py’

2. Command line

Command line argument format:

> “python dynamicmap.py 7 user password evnimageid wettzellmapid”
> Or
> “python dynamicmap.py 7” for default configuration

For example:

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 7 Admin zabbix 215 31
EVN global map created successfully
('Request 7 result: ', {u'jsonrpc': u'2.0', u'result': {u'sysmapids': [u'32']},
u'id': 8})
```

Result is the map ID of new created map for EVN global network(“evnmapid”).

Error analysis:

When image ID or map IDs are wrongly input, the result shows “request error”.

When login account is error, the result shows “Not authorised”.

When request number is wrongly input or missed, the result shows “request not exists”

6. Update maps and antenna button operations

Request 5 and 8 are used to update maps for Wettzell Station and EVN global network respectively, and antenna buttons can be therefore controlled.

For Request 5,

How to use:

1. Python IDE

- Update Zabbix system user account and password, url of Zabbix server
- Update image ID of Wettzell station, map ID of Wettzell station, host ID of twin telescopes at Wettzell station
- Update information of antenna/telescope/station/submap on the map
- Set request number (line 548) to 5
- Run ‘dynamicmap.py’

2. Command line

Command line argument format:

> “python dynamicmap.py 5 user password wettzellmapid wettzellimageid hostid1 hostid2”
> Or
> “python dynamicmap.py 5” for default configuration

How to define antenna button:

The antenna/telescope/station is defined as map element (element type: host, submap etc.) on the map. For the map of Wettzell station, individual antenna is shown as ‘host’ and can be modified through scripts of line 304 to 335.

```
304            "selements": [
305                {
306                    "selementid": "1",
307                    "elements": [
308                        {"hostid": hostid1}
309                    ],
310                    "elementtype": 0,                    # Type of map element: host
311                    "application": "Twin Telescope 1",   # application of the host
312                    "iconid_off": "143",                 # ID of the image used to display the element in default state.
313                    "iconid_disabled": "144",            # ID of the image used to display disabled map elements. Unused for image elements.
314                    "iconid_maintenance": "144",         # ID of the image used to display map elements in maintenance. Unused for image elements.
315                    "iconid_on": "144",                  # ID of the image used to display map elements with problems. Unused for image elements.
316                    "label": "Twin Telescope 1",
317                    "label_location": 0,
318                    "x": 220,                            # coordinate of telescope
319                    "y": 190
320                },
321                {
322                    "selementid": "2",
323                    "elements": [
324                        {"hostid": hostid2}
325                    ],
326                    "elementtype": 0,
327                    "application": "Twin Telescope 2",
328                    "iconid_off": "143",
329                    "iconid_disabled": "144",
330                    "iconid_maintenance": "144",
331                    "iconid_on": "144",
332                    "label": "Twin Telescope 2",
333                    "label_location": 0,
334                    "x": 268,
335                    "y": 255
336                },
337            ]
```

If an antenna is expected to be switched off, delete certain lines.

If an antenna is expected to be switched on, add certain lines.

More about element parameters, see developer manual.

For example:

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 5 Admin zabbix 31 214 10084 10084
Wettzell map updated successfully
('Request 5 result: ', {u'jsonrpc': u'2.0', u'result': {u'sysmapids': [31]}, u'i
d': 6})
```

Result is the map ID of updated map for Wettzell station.

Error analysis:

When image ID or host IDs are wrongly input, the result shows “request error”.

When login account is error, the result shows “Not authorised”.

When request number is wrongly input or missed, the result shows “request not exists”

For Request 8,

How to use:

1. Python IDE

- Update Zabbix system user account and password, url of Zabbix server
- Update map ID of EVN global network, image ID of EVN global network, map ID of Wettzell station
- Update information of antenna/telescope/station/submap on the map
- Set request number (line 548) to 8
- Run ‘dynamicmap.py’

2. Command line

Command line argument format:

> “python dynamicmap.py 8 user password evnmapid evnimageid wettzellmapid”
> Or
> “python dynamicmap.py 8” for default configuration

How to define antenna button:

The antenna/telescope/station is defined as map element (element type: host, submap etc.) on the map. For the map of EVN global network, individual station is shown as submap and can be modified through scripts of line 478 to 494.

```
478        "selements": [
479            {
480                "selementid": "1",
481                "elements": [
482                    {"sysmapid": submapid}
483                ],
484                "elementtype": 1,            # defines the map element: submap
485                "iconid_off": "144",         # ID of the image used to display the element in default state.
486                "iconid_disabled": "145",    # ID of the image used to display disabled map elements. Unused for image elements.
487                "iconid_maintenance": "145",# ID of the image used to display map elements in maintenance. Unused for image elements.
488                "iconid_on": "145",          # ID of the image used to display map elements with problems. Unused for image elements.
489                "label": "Wettzell",
490                "label_location": 0,
491                "x": 1008,                   # coordinate of telescopes
492                "y": 151
493            }
494        ]
```

If a station is expected to be switched off, delete certain lines.

If a station is expected to be switched on, add certain lines.

More about element parameters, see developer manual.

For example:

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 8 Admin zabbix 32 215 31
EVN global map updated successfully
('Request 8 result: ', {u'jsonrpc': u'2.0', u'result': {u'sysmapids': [32]}, u'i
d': 9})
```

Result is the map ID of updated map for EVN global network.

Error analysis:

When image ID or map IDs are wrongly input, the result shows “request error”.

When login account is error, the result shows “Not authorised”.

When request number is wrongly input or missed, the result shows “request not exists”

7. Delete maps

Request 6 and 9 are used to delete maps for Wettzell Station and EVN global network respectively.

For Request 6,

How to use:

1. Python IDE

- Update Zabbix system user account and password, url of Zabbix server
- Update map ID of Wettzell station
- Set request number (line 548) to 6
- Run ‘dynamicmap.py’

2. Command line

Command line argument format:

> “python dynamicmap.py 6 user password wettzellmapid”
> Or
> “python dynamicmap.py 6” for default configuration

For example:

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 6 Admin zabbix 31
Wettzell map deleted successfully
('Request 6 result: ', {u'jsonrpc': u'2.0', u'result': {u'sysmapids': [31]}, u'i
d': 7})
```

Result is the map ID of deleted map for Wettzell station.

Error analysis:

When image ID or host IDs are wrongly input, the result shows “request error”.

When login account is error, the result shows “Not authorised”.

When request number is wrongly input or missed, the result shows “request not exists”

For Request 9,

How to use:

1. Python IDE

- Update Zabbix system user account and password, url of Zabbix server
- Update map ID of EVN global network
- Set request number (line 548) to 9
- Run ‘dynamicmap.py’

2. Command line

Command line argument format:

> “python dynamicmap.py 9 user password evnmapid”
> Or
> “python dynamicmap.py 9” for default configuration

For example:

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 9 Admin zabbix 32
EVN global map deleted successfully
('Request 9 result: ', {u'jsonrpc': u'2.0', u'result': {u'sysmapids': [32]}, u'i
d': 10})
```

Result is the map ID of updated map for EVN global network.

Error analysis:

When image ID or map IDs are wrongly input, the result shows “request error”.

When login account is error, the result shows “Not authorised”.

When request number is wrongly input or missed, the result shows “request not exists”

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 10 Admin zabbix 32
request not exists . please check again.
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 Admin zabbix 32
request not exists . please check again.
```

Student Task 5 Dynamic ZABBIX Maps for JumpingJIVE

Developer manual

1. Introduction

1.1 Basics

The scripts are written in Python 3, consisting of 3 .py files:

‘dynamicmap.py’
‘login.py’
‘images.py’

Main functions shall be realized in ‘dynamicmap.py’. The other two work as modules.

1.2 Functions

The scripts are expected to realize following functions:

1. Get host information

2. Create images as background of maps

3. Create Wettzell station map and EVN global map

4. Determine station/antennas location on the map

5. Add or delete stations/antennas on the map

6. Update and delete maps

7. Possibility to switch on/off the antennas on the map

1.3 Command line argument

Except Python IDE, the scripts are expected to be called and run from Command Prompt by command line argument.

```
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 6 Admin zabbix 31
Wettzell map deleted successfully
('Request 6 result: ', {u'jsonrpc': u'2.0', u'result': {u'sysmapids': [31]}, u'i
d': 7})
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 9 Admin zabbix 32
EVN global map deleted successfully
('Request 9 result: ', {u'jsonrpc': u'2.0', u'result': {u'sysmapids': [32]}, u'i
d': 10})
jingyao@ThinkPad-Su-Ubuntu:~/JumpingJIVE non-sync/Pyscript$ python dynamicmap.py
 10 Admin zabbix 32
request not exists . please check again.
```

Fig.2 Command line argument in Ubuntu Terminal

1.4 Environment

Since Python 3 should be already installed in the Ubuntu Linux system, only few third-party Python packages (requests) are needed to be installed in extra.

To see which version of Python 3 you have installed, open a command prompt and run

```
$ python3 --version
```

If you are using Ubuntu 16.10 or newer, then you can easily install Python 3.6 with the following commands:

```
$ sudo apt-get update
$ sudo apt-get install python3.6
```

If you're using another version of Ubuntu (e.g. the latest LTS release), to install Python 3.6:

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt-get update
$ sudo apt-get install python3.6
```

Python 3.4 and later include pip by default. To see if pip is installed, open a command prompt and run

```
$ command -v pip
```

If no pip is installed, run

```
$ apt-get install pip
```

Then to install 'requests' package, run

```
$ pip install requests
```

2. Script structure

2.1 'dynamicmap.py'

1. Definition and login account

Login account of Zabbix system and IP address of Zabbix server, as well as other definitions are expected to be updated in this part. In addition, these parameters except server address can also be determined from command line argument, which is expected to have higher priority, which means the values in this part should be overwritten by command line arguments. Where should be paid attention is the server address can only be defined here.

```
# ==============================================================================
# Definitions and login account
# ==============================================================================
# Zabbix server address, only update here
url = 'http://192.168.1.105/zabbix/api_jsonrpc.php' # test Zabbix server on laptop

# Login information, update here if no extra input from command lines
user = "Admin"
password = "zabbix"

# system IDs of images or maps in Zabbix, update here if no extra input from command lines
wettzellimageid = 214
wettzellmapid = 33
evnimageid = 215
evnmapid = 34
hostid = 10084
hostid1 = 0
hostid2 = 0
hostname = "Zabbix server"

# images in BASE64 format
image4000_2000 = images.image4000_2000
image2000_1000 = images.image2000_1000
image1000_500 = images.image1000_500
imagewettzell = images.imagewettzell
```

Definitions are:

‘url’: Zabbix server address
‘user’: User account of Zabbix system
‘password’: User password of Zabbix system
‘wettzellimageid’: background image ID for Wettzell station map
‘wettzellmapid’: map ID of Wettzell station in Zabbix system
‘evnimageid’: background image ID for EVN global map
‘evnmapid’: map ID of EVN global network in Zabbix system
‘hostid’: host ID of telescope/antenna to be monitored
‘hostid1’: host ID of twin telescope 1 at Wettzell station
‘hostid2’: host ID of twin telescope 2 at Wettzell station
‘hostname’: name of host whose information is expected to be obtained

Also the images are introduced in the formation of BASE64 from ‘images.py’.

‘image4000_2000’: background image for EVN global map with size of 4000*2000
‘image2000_1000’: background image for EVN global map with size of 2000*1000
‘image1000_500’: background image for EVN global map with size of 1000*500
‘imagewettzell’: background image for Wettzell station map

2. Request 1 to 9

All of 9 requests are programmed as functions and thus are expected to run when it is called.

Details in Chapter 3 to 5 of this manual.

3. Main function

Command line arguments are expected to read in this part and overwrite default values.

After checking availability of input data, the requests should be run according to requirements, i.e. if there is argument, run it, while if no, run default.

```python
# ==============================================================================
# Main function
# ==============================================================================
def main():
    global user,password,request_number,hostid1,hostid2,wettzellimageid
    global evnimageid,wettzellmapid,evnmapid,hostname
    # login through command line
    if len(sys.argv) >= 4:
        user = sys.argv[2]
        password = sys.argv[3]

    # Selection of request to be deployed
    if len(sys.argv) == 1:
        request_number = 1 # defaut request to be deployed,
                           # update here if no extra input from command lines
    else:
        if sys.argv[1].isdigit():
            request_number = int(sys.argv[1])
        else:
            request_number = 0

    # deploy the request
    if request_number < 1 or request_number > 9:
        print("request not exists . please check again.")
    else:
        if request_number == 1:
            if len(sys.argv) == 5:
                hostname = int(sys.argv[4])
                request1(hostname)
            else:
                request1()
        if request_number == 2:
            request2()
        if request_number == 3:
            request3()
        if request_number == 4:
            if len(sys.argv) == 7:
                wettzellimageid = int(sys.argv[4])
                hostid1 = int(sys.argv[5])
                hostid2 = int(sys.argv[6])
                request4(wettzellimageid,hostid1,hostid2)
            else:
                request4()
```

4. Exception handling

This part is expected to deal with exceptions in case of the Python error handling with endless outputs of the whole call stack and line numbers.

```python
# ==============================================================================
# Exception handling
# ==============================================================================
if __name__ == '__main__':
    try:
        main()
    except BaseException:
        print ("request error. see error message.")
        exit(1)
```

2.2 'login.py'

This file is expect to deal with system login. Whenever it is called with the input of user and password, it should return a token(auth). Otherwise without input the default account is set to be 'Admin' and 'zabbix' for user name and password respectively.

Meanwhile, user does not need to update anything (user account or server address) in this file, but only operate in 'dynamicmap.py'.

1. system login

```python
def syslogin(user,passwd,url):
    URL = url

    LOGIN={
        "jsonrpc": "2.0",
        "method": "user.login",
        "params": {
            "user": user,
            "password": passwd
            },
        "id": 1,
        "auth": None
        }

    r = requests.post(url = URL, json = LOGIN, verify = False)
    data = r.json()
    auth = data.get('result')
    return auth
```

This part is expected to realize the login function. Input should be user name, password and URL of Zabbix server. These inputs are expected to be defined from 'dynamicmap.py' by arguments or default value.

2. Main function as exception handling

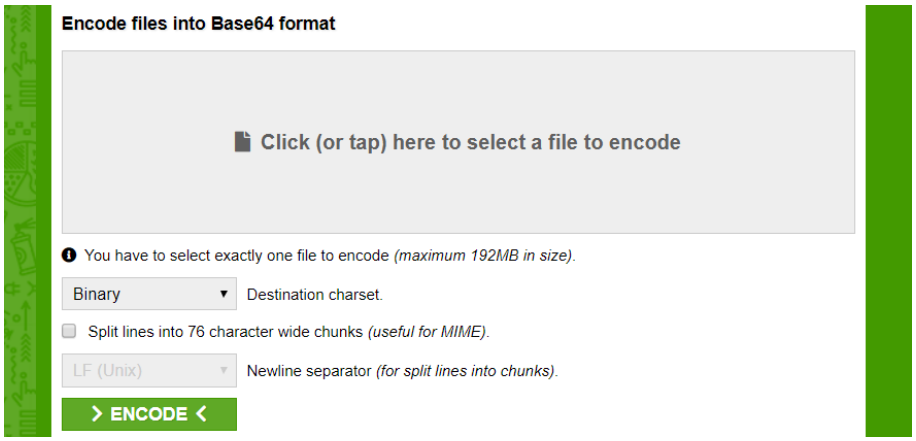
```python
def main(user = USER, passwd = PASSWORD, url = URL):
    try:
        authcode = syslogin(user,passwd,url)
    except BaseException:
        print ("Login error or could not connect to server at "+ URL)
        exit(1)
    return authcode


if __name__ == '__main__':
    main()
```

This part is expected to deal with exceptions in case of the Python error handling with endless outputs of the whole call stack and line numbers.

2.3 'images.py'

Images in formation of BASE64 are expected to be defined in this file and can be introduced to 'dynamicmap.py'. Currently there are 4 images

The BASE64 code can be obtained by an online tool 'BASE64 Decode and Encode'. The address is: https://www.base64encode.org/



How to use it:

- Select an image file to upload

- Choose ‘Binary’ as destination charset
- Click ‘Encode’ and get a ‘Success!’ result
- Click ‘CLICK OR TAP HERE’ to download the code.

3. Get host info

This chapter deals with function of getting host information. Related part of script is ‘Request 1: login and get host info’ from line 78 to 118.

```
78  # ==============================================================================
79  # Request 1: login and get host info
80  # ==============================================================================
81  def request1(host=hostname):
82      # definitions
83      global hostid
84
85      # get sessionid
86      token = login.main()
87
88      # request content
89      post_data = {
90          "jsonrpc": "2.0",
91          "method": "host.get",
92          "params": {
93              "filter": {
94                  "host": [
95                      host
96                  ]
97              },
98              "output": [
99                  "hostid",
100                 "host"
101             ],
102             "selectInterfaces": [
103                 "interfaceid",
104                 "ip"
105             ]
106         },
107         "id": 2,
108         "auth": token
109     }
110     r = requests.post(url, json = post_data, verify = False)
111     zabbix_ret = r.json()
112
113     if not zabbix_ret.__contains__('result'):
114         print ('Request 2 image creates error:',zabbix_ret.get('error')['data'])
115     else:
116         hostid = zabbix_ret['result'][0]['hostid']
117         print ('login succeeds. Your host is', zabbix_ret['result'][0]['host'],'w
118                                                 # get the current server host information. wa
```

Input is expected to be the host to be monitored. Default is current server.

Firstly ‘login.py’ is called to get a login token, which is supposed to be referred for any request.

Variable of ‘post_data’ defines the parameters of request.

Here lies host name, which is supposed to be updated by user, ‘output’, which includes host ID and host name and shows in the request result, as well as IP address.

After checking if the request runs successfully or not, the result is expected to be printed.

4. Image creation

This chapter deals with the function of background image creation. Related part of script is ‘Request 2: create images as background for Wettzell Station’ from line 121 to 151 and ‘Request 3: create images as background for global network’ from line 153 to 183.

```python
# ==============================================================================
# Request 2: create images as background for Wettzell Station
# ==============================================================================
def request2():
    # definitions
    global wettzellimageid

    # get sessionid
    token = login.main(user,password,url)

    # request content
    post_data = {
        "jsonrpc": "2.0",
        "method": "image.create",
        "params": {
            "imagetype": 2,             # type of background image
            "name": "Wettzell station image",
            "image": imagewettzell
        },
        "auth": token,
        "id": 3
    }
    r = requests.post(url, json = post_data, verify = False)
    zabbix_ret = r.json()

    if not zabbix_ret.__contains__('result'):
        print ('Request 2 image creates error:',zabbix_ret.get('error')['data'])
    else:
        wettzellimageid = zabbix_ret.get('result')['imageids'][0]
        print("Wettzell image created successfully")
        print("Request 2 result: ", zabbix_ret)
```

```python
# ==============================================================================
# Request 3: create images as background for global network
# ==============================================================================
def request3():
    # definitions
    global evnimageid

    # get sessionid
    token = login.main(user,password,url)

    # request content
    post_data = {
        "jsonrpc": "2.0",
        "method": "image.create",
        "params": {
            "imagetype": 2,             # type of background image
            "name": "EVN global map",
            "image": image2000_1000
        },
        "auth": token,
        "id": 4
    }
    r = requests.post(url, json = post_data, verify = False)
    zabbix_ret = r.json()

    if not zabbix_ret.__contains__('result'):
        print ("Request 3 image creates error:",zabbix_ret.get('error')['data'])
    else:
        evnimageid = zabbix_ret.get('result')['imageids'][0]
        print("Global map image created successfully")
        print("Request 3 result: ", zabbix_ret)
```

These two parts have same structure.

There is no input required when calling this function.

Firstly ‘login.py’ is called to get a login token, which is supposed to be referred for any request.

Variable of ‘post_data’ defines the parameters of request.

Here lies ‘imagetype’, which defines the type of image in Zabbix system, here setting ‘2’ as background image, ‘name’, which defined the name of image, as well as ‘image’, which should be BASE64 code. Default image is background image for Wettzell station map or background image for global map with size of 2000*1000.

After checking if the request runs successfully or not, the result is expected to be printed. Result is expected to include the ID of stored image, which is named as a variable of ‘wettzellimageid’ or ‘evnimageid’. Otherwise the result show error message.

5. Map creation, updating and deletion

This chapter deals with the function of map creation, updating and deletion. Related part of script is ‘Request 4: create map of Wettzell Station’ from line 185 to 269, ‘Request 5: update map of Wettzell Station’ from line 271 to 351, ‘Request 6: delete map of Wettzell station’ from line 353 to 377, ‘Request 7: create map of EVN global network’ from line 379 to 444, ‘Request 8: update map of EVN global network’ from 446 to 508 and ‘Request 9: delete map of EVN global networks’ from line 510 to 535.

Basic structure is as follows:

Firstly is input setting.

- For map creation, inputs required should be background image ID and IDs of the map elements, which are supposed to be ‘hostid’ for host(antenna/telescope) and ‘sysmapids’ for submap (station with separate map, e.g. Wettzell Station) when calling this function.
- For map updating, inputs required has additional ID of current map, i.e. ‘wettzellmapid’ for Wettzell Station map and ‘envmapid’ for EVN global map.
- For map deletion, input required is supposed to be only the ID of current map.

Then ‘login.py’ is called to get a login token, which is supposed to be referred for any request.

Variable of ‘post_data’ defines the parameters of request.

For map creation and map updating, ‘param’ defines all the properties of map to be created or to be updated. Map updating has an additional parameter of ‘sysmapid’.

```
196     # request content
197     post_data = {
198         "jsonrpc": "2.0",
199         "method": "map.create",
200         "params": {
201             "name": "Wettzell Station",
202             "width": 400,
203             "height": 300,
204             "backgroundid": imageid,        # image ID obtained from creation or input
205             "expandproblem": 0,             # the problem trigger will be displayed for elements wit
206             "label_type_map": 2,            # Map element label type: default delement name
207             "grid_align": 0,                # disable grid aligning to fix station coordinate
208             "highlight": 1,                 # icon highlighting is enabled
209             "label_location": 0,            # Location of the map element label: bottom as default
210             "label_type_host": 0,           # Custom label for host elements.
211             "shapes": [
212                 {                           # defines the title of map
213                     "type": 0,
214                     "x": 150,
215                     "y": 30,
216                     "width": 100,
217                     "height": 10,
218                     "text": "{MAP.NAME}",
219                     "font": 4,
220                     "font_size": 11
221                 }
222             ],
223             "selements": [
256             ]
257         },
258         "auth": token,
259         "id": 5
260     }
```

- "sysmapid": current map ID, i.e.’wettzellmapid’ or ‘evnmapid’. This ID is obtained from its creation result.

- "name": "Wettzell Station": name of map
- "width": width of map
- "height": height of map
- "backgroundid": defined from variable ‘imageid’, background image ID
- "expandproblem": the problem trigger will be displayed for elements with a single problem. ‘0’ means always displaying the number of problems
- "label_type_map": Map element label type: ‘2’ means default is element name
- "grid_align": ‘0’ means disable grid aligning to fix station coordinate
- "highlight": ‘1’ means icon highlighting is enabled
- "label_location": Location of the map element label: ‘0’ means bottom as default
- "label_type_host": Custom label for host elements.
- "shapes": defines the title of map
- "selements": defines the map elements, detailed description in next part

For map deletion, ‘param’ only includes ID of map to be deleted.

```
# ==============================================================================
# Request 6: delete map of Wettzell station
# ==============================================================================
def request6(mapid=wettzellmapid):
    # get sessionid
    token = login.main(user,password,url)

    # request content
    post_data = {
        "jsonrpc": "2.0",
        "method": "map.delete",
        "params": [
            mapid # here is the id of map to be deleted
        ],
        "auth": token,
        "id": 7
    }
    r = requests.post(url, json = post_data, verify = False)
    zabbix_ret = r.json()

    if not zabbix_ret.__contains__('result'):
        print ("Request 6 map deletes error:",zabbix_ret.get('error')['data'])
    else:
        print("Wettzell map deleted successfully")
        print("Request 6 result: ", zabbix_ret)
```

After checking if the request runs successfully or not, the result is expected to be printed. Result is expected to include the ID of stored image, which is named as a variable of ‘wettzellimageid’ or ‘evnimageid’. Otherwise the result show error message.

Important is the map element definition, which determines what is going to show on the map and thus it is possible to switch an antenna on or off.

Basic idea is simply adding or deleting certain lines when an antenna/telescope/station is expected to switch on or off. Following is the parameters of map elements:

- "selementid": ID of map element
- "elements": map element, host or submap, value should be host ID or map ID, such as {"hostid": hostid1} or {"sysmapid": submapid}
- "elementtype": Type of map element: ‘0’ as host, ‘1’ as submap
- "application": application of the host
- "iconid_off": ID of the image used to display the element in default state.
- "iconid_disabled": ID of the image used to display disabled map elements. Unused for image elements.
- "iconid_maintenance": ID of the image used to display map elements in maintenance. Unused for image elements.

- "iconid_on": ID of the image used to display map elements with problems. Unused for image elements.
- "label": label of element
- "label_location": location of label
- "x", "y": coordinate of antenna button on the map

Following is code for an element type of ‘host’, which represents antenna/telescope, from Request 5:

```
323                 {
324                     "selementid": "2",
325                     "elements": [
326                         {"hostid": hostid2}
327                     ],
328                     "elementtype": 0,
329                     "application": "Twin Telescope 2",
330                     "iconid_off": "143",
331                     "iconid_disabled": "144",
332                     "iconid_maintenance": "144",
333                     "iconid_on": "144",
334                     "label": "Twin Telescope 2",
335                     "label_location": 0,
336                     "x": 268,
337                     "y": 255
338                 },
```

Following is code for an element type of ‘submap’, which represents station with separate map, from Request 8:

```
416                 {
417                     "selementid": "1",
418                     "elements": [
419                         {"sysmapid": submapid}
420                     ],
421                     "elementtype": 1,
422                     "iconid_off": "144",
423                     "iconid_disabled": "145",
424                     "iconid_maintenance": "145",
425                     "iconid_on": "145",
426                     "label": "Wettzell",
427                     "label_location": 0,
428                     "x": 1008,
429                     "y": 151
430                 }
```