py-kms Documentation

SystemRage

Mar 23, 2024

CONTENTS

Contents:

CHAPTER

ONE

CONTRIBUTING

You want to improve this project? Awesome! But before you write or modify the existing source code, please note the following guideline:

- Always make sure to add your changes to the wiki.
- 8-space indentation without tabs.
- Docstrings as this:

```
    """ This is single line docstring. """
    """ This is a
    """ multiline comment.
```

- Wrap lines only if really long (it does not matter 79 chars return)
- For the rest a bit as it comes with a look at PEP8 :)

CHAPTER

TWO

DOCUMENTATION

What follows are some detailed explanations how the KMS infrastructure works.

2.1 Understanding Key Management Service

KMS activates Microsoft products on a local network, eliminating the need for individual computers to connect to Microsoft. To do this, KMS uses a client–server topology. A KMS client locates a KMS server by using DNS or a static configuration, then contact it by using Remote Procedure Call (RPC) and tries to activate against it. KMS can activate both physical computers and virtual machines, but a network must meet or exceed the activation threshold (minimum number of computers that KMS requires) of 25. For activation, KMS clients on the network need to install a KMS client key (General Volume License Key, GVLK), so the product no longer asks Microsoft server but a user–defined server (the KMS server) which usually resides in a company's intranet.

py-kms is a free open source KMS server emulator written in Python, while Microsoft gives their KMS server only to corporations that signed a Select contract. Furthermore *py-kms* never refuses activation since it is without restrictions, while the Microsoft KMS server only activates the products the customer has paid for. *py-kms* supports KMS protocol versions 4, 5 and 6.

Although *py-kms* does neither require an activation key nor any payment, it is not meant to run illegal copies of Windows. Its purpose is to ensure that owners of legal copies can use their software without restrictions, e.g. if you buy a new computer or motherboard and your key will be refused activation from Microsoft servers due to hardware changes.

Activation with *py-kms* is achieved with the following steps:

1. Run *py-kms* on a computer in the network (this is KMS server or local host).
2. Install the product on client (or said remote host, which is the computer sending data to local host) and enter the GVLK.
3. Configure the client to use the KMS server.

Note that KMS activations are only valid for 180 days, the activation validity interval, or 30 to 45 days with consumer-only products. To remain activated, KMS client computers must renew their activation by connecting to the KMS server at least once every 180 days. For this to work, you have to should ensure that a KMS server is always reachable for all clients on the network. Also remember **you can't activate Windows 8.1 (and above) on a KMS server hosted on the same machine** (the KMS server must be a different computer than the client).

2.1.1 About GVLK keys

The GVLK keys for products sold via volume license contracts (renewal every 180 days) are published on Microsoft's Technet web site.

- Windows: https://technet.microsoft.com/en-us/library/jj612867.aspx
- Office 2010: https://technet.microsoft.com/en-us/library/ee624355(v=office.14).aspx#section2_3
- Office 2013: https://technet.microsoft.com/en-us/library/dn385360.aspx
- Office 2016: https://technet.microsoft.com/en-en/library/dn385360(v=office.16).aspx

There are also not official keys for consumer-only versions of Windows that require activation renewal every 45 days (Windows 8.1) or 30 days (Windows 8). A more complete and well defined list is available *here*.

2.1.2 SLMGR and OSPP commands

The software License Manager (`slmgr.vbs`) is a Visual Basic script used to configure and retrieve Volume Activation information. The script can be run locally or remotely on the target computer, using the Windows-based script host (`wscript.exe`) or the command-based script host (`cscript.exe`) - administrators can specify which script engine to use. If no script engine is specified, *SLMGR* runs using the default script engine (it is recommended to utilize the `cscript.exe` script engine that resides in the system32 directory). The Software Licensing Service must be restarted for any changes to take effect. To restart it, the Microsoft Management Console (MMC) Services can be used or running the following command:

```
net stop sppsvc && net start sppsvc
```

The *SLMGR* requires at least one parameter. If the script is run without any parameters, it displays help information. The general syntax of `slmgr.vbs` is as follows (using the `cscript.exe` as the script engine):

```
cscript slmgr.vbs /parameter
cscript slmgr.vbs [ComputerName] [User] [Password] [Option]
```

Where command line options are:

```
[ComputerName]  Name of a remote computer (default is local computer).
[User]          Account with the required privilege on the remote computer.
[Password]      Password for the account with required privileges on the remote compute.
[Option]        Options are shown in the table below.
```

SLMGR

Following tables lists *SLMGR* more relevant options and a brief description of each. Most of the parameters configure the KMS host.

OSPP

The Office Software Protection Platform script (`ospp.vbs`) can help you to configure and test volume license editions of Office client products. You must open a command prompt by using administrator permissions and navigate to the folder that contains the mentioned script. The script is located in the folder of the Office installation (use `\Office14` for Office 2010, `\Office15` for Office 2013 and `\Office16` for Office 2016): `%installdir%\Program Files\Microsoft Office\Office15`. If you are running a 32-bit Office on a 64-bit operating system, the script is located in the folder: `%installdir%\Program Files (x86)\Microsoft Office\Office15`.

Running *OSPP* requires the `cscript.exe` script engine. To see the help file, type the following command, and then press ENTER:

```
cscript ospp.vbs /?
```

The general syntax is as follows:

```
cscript ospp.vbs [Option:Value] [ComputerName] [User] [Password]
```

Where command line options are:

```
[Option:Value]  Specifies the option and Value to use to activate a product, install or␣
↪uninstall a product key, install and display license information, set KMS host name␣
↪and port, and remove KMS host. The options and values are listed in the tables below.
[ComputerName]  Name of the remote computer. If a computer name is not provided, the␣
↪local computer is used.
[User]          Account that has the required permission on the remote computer.
[Password]      Password for the account. If a user account and password are not␣
↪provided, the current credentials are used.
```

2.2 Supported Products

Note that it is possible to activate all versions in the VL (Volume License) channel, so long as you provide the proper key to let Windows know that it should be activating against a KMS server. KMS activation can't be used for Retail channel products, however you can install a VL product key specific to your edition of Windows even if it was installed as Retail. This effectively converts Retail installation to VL channel and will allow you to activate from a KMS server. **However, this is not valid for Office's products**, so Office, Project and Visio must be always volume license versions. Newer version may work as long as the KMS protocol does not change...

2.3 Further References

- [1] https://forums.mydigitallife.net/threads/emulated-kms-servers-on-non-windows-platforms.50234
- [2] https://forums.mydigitallife.net/threads/discussion-microsoft-office-2019.75232
- [3] https://forums.mydigitallife.net/threads/miscellaneous-kms-related-developments.52594
- [4] https://forums.mydigitallife.net/threads/kms-activate-windows-8-1-en-pro-and-office-2013.49686
- [5] https://github.com/myanaloglife/py-kms
- [6] https://github.com/Wind4/vlmcsd
- [7] https://github.com/ThunderEX/py-kms
- [8] https://github.com/CNMan/balala/blob/master/pkconfig.csv

- [9] http://www.level7techgroup.com/docs/kms_overview.pdf
- [10] https://www.dell.com/support/article/it/it/itbsdt1/sln266176/windows-server-using-the-key-management-service-kms-for-activation-of-volume-licensed-systems?lang=en
- [11] https://social.technet.microsoft.com/Forums/en-US/c3331743-cba2-4d92-88aa-9633ac74793a/office-2010-kms-current-count-remain-at-10?forum=officesetupdeployprevious
- [12] https://betawiki.net/wiki/Microsoft_Windows
- [13] https://thecollectionbook.info/builds/windows
- [14] https://www.betaarchive.com/forum/viewtopic.php%3Ft%3D29131+&cd=10&hl=it&ct=clnk&gl=it
- [15] https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=12&cad=rja&uact=8&ved=2ahUKEwjmvZihtOHeA by-step%2520guide.doc&usg=AOvVaw3kqhCu3xT-3r416DRGUUs_
- [16] https://www.itprotoday.com/windows-78/volume-activation-server-2008
- [17] https://docs.microsoft.com/en-us/windows-server/get-started-19/activation-19
- [18] https://docs.microsoft.com/en-us/windows-server/get-started/windows-server-release-info
- [19] https://support.microsoft.com/en-us/help/13853/windows-lifecycle-fact-sheet

CHAPTER

THREE

GETTING STARTED

What follows are some guides how to start the `pykms_Server.py` script, which provides the emulated server.

3.1 Running as a service

You can simply manage a daemon that runs as a background process. This can be achieved by using any of the notes below or by writing your own solution.

3.1.1 Docker

If you wish to get *py-kms* just up and running without installing any dependencies or writing own scripts: Just use Docker ! Docker also solves problems regarding the explicit IPv4 and IPv6 usage (it just supports both). The following command will download, "install" and start *py-kms* and also keep it alive after any service disruption.

```
docker run -d --name py-kms --restart always -p 1688:1688 -v /etc/localtime:/etc/
↪localtime:ro ghcr.io/py-kms-organization/py-kms
```

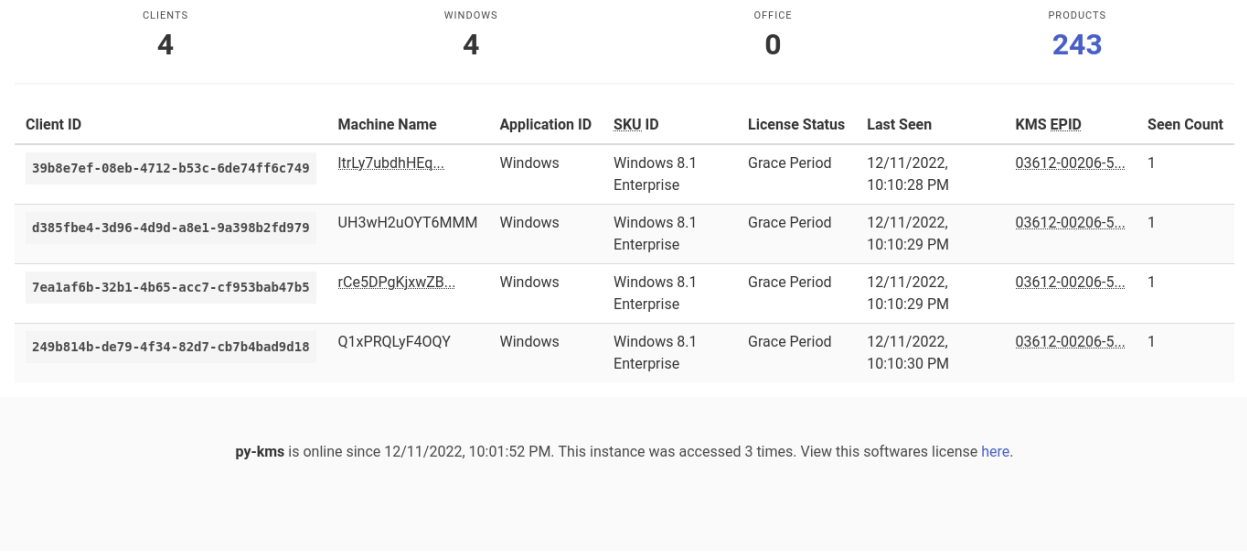
If you just want to use the image and don't want to build them yourself, you can always use the official image at the GitHub Container Registry (`ghcr.io/py-kms-organization/py-kms`). To ensure that you are using always the latest version you should check something like watchtower out!

Tags

There are currently three tags of the image available (select one just by appending `:<tag>` to the image from above):

- `latest`, currently the same like `minimal`.
- `minimal`, which is based on the python3 minimal configuration of py-kms. *This tag does NOT include* `sqlite` *support !*
- `python3`, which is fully configurable and equipped with `sqlite` support and a web-interface (make sure to expose port `8080`) for management.

Wait… Web-interface? Yes! `py-kms` now comes with a simple web-ui to let you browse the known clients or its supported products. In case you wonder, here is a screenshot of the web-ui (*note that this screenshot may not reflect the current state of the ui*):



CLIENTS	WINDOWS	OFFICE	PRODUCTS
4	4	0	243

Client ID	Machine Name	Application ID	SKU ID	License Status	Last Seen	KMS EPID	Seen Count
39b8e7ef-08eb-4712-b53c-6de74ff6c749	ItrLy7ubdhHEq...	Windows	Windows 8.1 Enterprise	Grace Period	12/11/2022, 10:10:28 PM	03612-00206-5...	1
d385fbe4-3d96-4d9d-a8e1-9a398b2fd979	UH3wH2uOYT6MMM	Windows	Windows 8.1 Enterprise	Grace Period	12/11/2022, 10:10:29 PM	03612-00206-5...	1
7ea1af6b-32b1-4b65-acc7-cf953bab47b5	rCe5DPgKjxwZB...	Windows	Windows 8.1 Enterprise	Grace Period	12/11/2022, 10:10:29 PM	03612-00206-5...	1
249b814b-de79-4f34-82d7-cb7b4bad9d18	Q1xPRQLyF4OQY	Windows	Windows 8.1 Enterprise	Grace Period	12/11/2022, 10:10:30 PM	03612-00206-5...	1

Architectures

There are currently the following architectures available (if you need an other, feel free to open an issue):

- `amd64`
- `arm32v6` Raspberry PI 1 (A, A+, B, B+, Zero)
- `arm32v7` Raspberry PI 2 (B)
- `arm64v8` Raspberry PI 2 (B v1.2), Raspberry PI 3 (A+, B, B+), Raspberry PI 4 (B)

Please note that any architecture other than the classic `amd64` *is slightly bigger (~4 MB), caused by the use of qemu during building.*

Docker Compose

You can use `docker-compose` instead of building and running the Dockerfile, so you do not need to respecify your settings again and again. The following Docker Compose file will deploy the `latest` image with the log into your local directory. Make sure to take a look into the `entrypoint.py` script to see all supported variable mappings!

```
version: '3'

services:
  kms:
    image: ghcr.io/py-kms-organization/py-kms:python3
    ports:
      - 1688:1688
      - 8080:8080
    environment:
      IP: "::"
      HWID: RANDOM
      LOGLEVEL: INFO
    restart: always
    volumes:
```

(continues on next page)

(continued from previous page)

```
      - ./db:/home/py-kms/db
      - /etc/localtime:/etc/localtime:ro
```

Parameters

Below is a little bit more extended run command, detailing all the different supported environment variables to set. For further reference see the *start parameters* for the docker environment.

```
docker run -it -d --name py3-kms \
    -p 8080:8080 \
    -p 1688:1688 \
    -v /etc/localtime:/etc/localtime:ro \
    --restart unless-stopped ghcr.io/py-kms-organization/py-kms:[TAG]
```

You can omit the `-p 8080:8080` option if you plan to use the `minimal` or `latest` image, which does not include the `sqlite` module support.

3.1.2 Systemd

If you are running a Linux distro using `systemd`, create the file: `sudo nano /etc/systemd/system/py3-kms.service`, then add the following (change it where needed) and save:

```
[Unit]
Description=py3-kms
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=1
KillMode=process
User=root
ExecStart=/usr/bin/python3 </path/to/your/pykms/files/folder>/py-kms/pykms_Server.py ::␣
↪1688 -V DEBUG -F </path/to/your/log/files/folder>/pykms_logserver.log

[Install]
WantedBy=multi-user.target
```

Check syntax with `sudo systemd-analyze verify py3-kms.service`, correct file permission (if needed) `sudo chmod 644 /etc/systemd/system/py3-kms.service`, then reload systemd manager configuration `sudo systemctl daemon-reload`, start the daemon `sudo systemctl start py3-kms.service` and view its status `sudo systemctl status py3-kms.service`. Check if daemon is correctly running with `cat </path/to/your/log/files/folder>/pykms_logserver.log`. Finally a few generic commands useful for interact with your daemon here.

3.1.3 Upstart (deprecated)

If you are running a Linux distro using `upstart` (deprecated), create the file: `sudo nano /etc/init/py3-kms.conf`, then add the following (change it where needed) and save:

```
description "py3-kms"
author "SystemRage"
env PYTHONPATH=/usr/bin
env PYKMSPATH=</path/to/your/pykms/files/folder>/py-kms
env LOGPATH=</path/to/your/log/files/folder>/pykms_logserver.log
start on runlevel [2345]
stop on runlevel [016]
exec $PYTHONPATH/python3 $PYKMSPATH/pykms_Server.py :: 1688 -V DEBUG -F $LOGPATH
respawn
```

Check syntax with `sudo init-checkconf -d /etc/init/py3-kms.conf`, then reload upstart to recognise this process `sudo initctl reload-configuration`. Now start the service `sudo start py3-kms`, and you can see the logfile stating that your daemon is running: `cat </path/to/your/log/files/folder>/pykms_logserver.log`. Finally a few generic commands useful for interact with your daemon here.

3.1.4 Windows

If you are using Windows, to run `pykms_Server.py` as service you need to install pywin32, then you can create a file for example named `kms-winservice.py` and put into it this code:

```
import win32serviceutil
import win32service
import win32event
import servicemanager
import socket
import subprocess

class AppServerSvc (win32serviceutil.ServiceFramework):
    _svc_name_ = "py-kms"
    _svc_display_name_ = "py-kms"
    _proc = None
    _cmd = ["C:\Windows\Python27\python.exe", "C:\Windows\Python27\py-kms\pykms_Server.py
↪"] # UPDATE THIS - because Python 2.7 is end of life and you will use other parameters␣
↪anyway

    def __init__(self,args):
        win32serviceutil.ServiceFramework.__init__(self,args)
        self.hWaitStop = win32event.CreateEvent(None,0,0,None)
        socket.setdefaulttimeout(60)

    def SvcStop(self):
        self.killproc()
        self.ReportServiceStatus(win32service.SERVICE_STOP_PENDING)
        win32event.SetEvent(self.hWaitStop)

    def SvcDoRun(self):
        servicemanager.LogMsg(servicemanager.EVENTLOG_INFORMATION_TYPE,
                              servicemanager.PYS_SERVICE_STARTED,
```

(continues on next page)

(continued from previous page)

```
                                (self._svc_name_,''))
        self.main()

    def main(self):
        self._proc = subprocess.Popen(self._cmd)
        self._proc.wait()

    def killproc(self):
        self._proc.kill()

if __name__ == '__main__':
    win32serviceutil.HandleCommandLine(AppServerSvc)
```

Now in a command prompt type `C:\Windows\Python27\python.exe kms-winservice.py install` to install the service. Display all the services with `services.msc` and find the service associated with *py-kms*, change the startup type from `manual` to `auto`. Finally `Start` the service. If this approach fails, you can try to use Non-Sucking Service Manager or Task Scheduler as described here.

3.1.5 Other Platforms

They might be useful to you:

- FreeNAS
- FreeBSD

3.2 Manual Execution

3.2.1 Dependencies

- Python 3.x.
- If the `tzlocal` module is installed, the "Request Time" in the verbose output will be converted into local time. Otherwise, it will be in UTC.
- It can use the `sqlite3` module, storing activation data in a database so it can be recalled again.
- Installation example on Ubuntu / Mint (`requirements.txt` is from the sources):
  - `sudo apt-get update`
  - `sudo apt-get install python3-pip`
  - `pip3 install -r requirements.txt` (on Ubuntu Server 22, you'll need `pysqlite3-binary` - see this issue)

3.2.2 Startup

A Linux user with `ip addr` command can get his KMS IP (Windows users can try `ipconfig /all`).

```
user@host ~ $ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen␣
↪1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp6s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group␣
↪default qlen 1000
    link/ether **:**:**:**:**:** brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.102/24 brd 192.168.1.255 scope global dynamic noprefixroute enp6s0
       valid_lft 860084sec preferred_lft 860084sec
    inet6 ****:****:****:****:****:****:****:****/64 scope global dynamic noprefixroute
       valid_lft 6653sec preferred_lft 3052sec
    inet6 ****::****:****:****:****/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

In the example above is `192.168.1.102` the ip we want to listen on, so it is this command (**note you can omit the ip AND port specification if you just wish to listen on all interfaces with port 1688**):

```
user@host ~/path/to/folder/py-kms $ python3 pykms_Server.py 192.168.1.102 1688
```

To stop `pykms_Server.py`, in the same bash window where code running, simply press `CTRL+C`. Alternatively, in a new bash window, use `kill <pid>` command (you can type `ps aux` first and have the process) or `killall <name_of_server>`.

3.2.3 Quick Guide

The following are just some brief notes about parameters handling. For a more detailed description see *here*.

- To generate a random HWID use `-w` option: `python3 pykms_Server.py -w RANDOM`.
- To get the HWID from any server use the client, for example type: `python3 pykms_Client.py :: 1688 -m Windows8.1 -V INFO`.
- To change your logfile path use `-F` option, for example: `python3 pykms_Server.py -F /path/to/your/logfile.log -V DEBUG`.
- To view a minimal set of logging information use `-V MININFO` option, for example: `python3 pykms_Server.py -F /path/to/your/logfile.log -V MININFO`.
- To redirect logging on stdout use `-F STDOUT` option, for example: `python3 pykms_Server.py -F STDOUT -V DEBUG`.
- You can create logfile and view logging information on stdout at the same time with `-F FILESTDOUT` option, for example: `python3 pykms_Server.py -F FILESTDOUT /path/to/your/logfile.log -V DEBUG`.
- With `-F STDOUTOFF` you disable all stdout messages (but a logfile will be created), for example: `python3 pykms_Server.py -F STDOUTOFF /path/to/your/logfile.log -V DEBUG`.
- With `-F FILEOFF` you disable logfile creation.
- Select timeout (seconds) for py-kms with `-t0` option, for example `python3 pykms_Server.py -t0 10`.

- Option `-y` enables printing asynchronously of messages (pretty / logging).

CHAPTER

FOUR

GVLK KEYS

These are keys, which can be used to activate a product with *py-kms* (note this keys are provided officially by Microsoft). *py-kms* will not reject any of your keys, instead the product will often revalidate the given key - and sometimes even reject it by itself (often due too many uses - in that case try to use an other one).

4.1 Windows

4.1.1 Windows Server 2022

4.1.2 Windows Server 2019

4.1.3 Windows Server 2016

4.1.4 Windows 10 & Windows 11

4.1.5 Windows Server 2012 R2

4.1.6 Windows 8.1

4.1.7 Windows Server 2012

4.1.8 Windows 8

4.1.9 Windows Server 2008 R2

4.1.10 Windows 7

4.1.11 Windows Server 2008

4.1.12 Windows Vista

4.1.13 Windows Previews

4.2 Office

4.2.1 Office 2021

4.2.2 Office 2019

4.2.3 Office 2016

4.2.4 Office 2013

CHAPTER

FIVE

TROUBLESHOOTING

Something does not work as expected ? *Before* you open an issue, please make sure to at least follow these steps to further diagnose or even resolve your problem. If you not follow this, do not expect that we can or want to help you!

- Are you activating a legit Windows copy checked with `sha256`, `md5` or is it maybe a warez torrent version ?
- Did you tried a clean installation (format all) ? You skipped entering any key during installation, turning off internet connection, first activating and then updating Windows (and eventually later upgrading) ?
- Are you activating Windows or Office on a different machine (physical or virtual) where py-kms runs?
- Have you installed all latest packages ? Especially before upgrading ? Are you upgrading using the "Update Assistant"/"Media Creation" tool to switch from Windows 7 / 8 / 8.1 to 10 (for me has always worked) ?
- If isn't a clean install, so far as you have kept activated your Windows copy ? Have you used some other activator (maybe not trusted) that injects or changes .dll files and therefore may have corrupted something ?
- Have you forgot to reactivate at least once before 180 (45 or 30, depending on your version) days ?
- Is your system very tweaked with some service disabled (have you used O&O Shutup 10 or similar tools) ?
- Have you disabled (or created an exception for) ALL firewalls (Public/Private/Office) / antivirus (Windows defender, etc..), server-side AND client-side ?
- Have you already activated with a OEM/Retail/other license and now you want to activate using `py-kms` ? So, have you switched to volume channel with appropriate *GVLK* ? Make sure you first deleted the previous key (example: #24 (comment)) ?
- Are you running the commands using the **elevated** command prompt ?
- Are you connecting correctly to your `py-kms` server instance ?
- Have you tried to fix with "Windows Troubleshoot", `sfc /scannow` or other strange Windows tools ?
- If you activated successfully with `py-kms` other Windows stuff, consider it could be an error specific for your PC (so you may need a scented clean installation) ?
- Is your `py-kms` really running ? Already tried to enable debug logs ?
- Did you already searched for your issue here ?
- Are you running the latest version of `py-kms` ?
- For Office: Did you made sure to use a Office **with** GLVK support ?!
- You found a real bug ? Could you maybe make our life's easier and describe what goes wrong **and** also provide some information about your environment (OS, Python-Version, Docker, Commit-Hashes, Network-Setup) ?
- When you post logs: Please remove personal information (replace IPs with something like `[IP_ADDRESS_A]`)...

If you have already thought about all of this, your last hope to solve your problem is reading some verse of the Holy Bible of activations: "MDL forums" - otherwise "I don't know !", but try open up an issue anyways :)

CHAPTER

SIX

USAGE

6.1 Start Parameters

6.1.1 pykms_Server.py

Follows a list of usable parameters:

```
ip <IPADDRESS>
```

Instructs py-kms to listen on *IPADDRESS* (can be an hostname too). If this option is not specified, *IPADDRESS* `::` is used.

```
port <PORT>
```

Define TCP *PORT* the KMS service is listening on. Default is 1688.

```
-e or --epid <EPID>
```

Enhanced Privacy ID (*EPID*) is a cryptographic scheme for providing anonymous signatures. Use *EPID* as Windows *EPID*. If no *EPID* is specified, a random one will be generated.

```
-l or --lcid <LCID>
```

Specify the *LCID* part of the *EPID*. If an *EPID* is manually specified, this setting is ignored. Default is 1033 (English - US). The Language Code Identifier (*LCID*) describes localizable information in Windows. This structure is used to identify specific languages for the purpose of customizing software for particular languages and cultures. For example, it can specify the way dates, times, and numbers are formatted as strings. It can also specify paper sizes and preferred sort order based on language elements. The *LCID* must be specified as a decimal number (example: 1049 for "Russian - Russia"). By default py-kms generates a valid locale ID but this may lead to a value which is unlikely to occur in your country. You may want to select the locale ID of your country instead. See here for a list of valid *LCIDs*.

```
-w or --hwid <HWID>
```

Use specified *HWID* for all products. Use `-w RANDOM` to generate a random HWID. Default is random. Hardware Identification is a security measure used by Microsoft upon the activation of the Windows operating system. As part of the Product Activation system, a unique HWID number is generated when the operating system is first installed. The *HWID* identifies the hardware components that the system is utilizing, and this number is communicated to Microsoft. Every 10 days and at every reboot the operating system will generate another *HWID* number and compare it to the original to make sure that the operating system is still running on the same device. If the two *HWID* numbers differ too much then the operating system will shut down until Microsoft reactivates the product. The theory behind *HWID* is to ensure that

the operating system is not being used on any device other than the one for which it was purchased and registered. HWID must be an 16-character string of hex characters that are interpreted as a series of 8 bytes (big endian).

```
-c or --client-count <CLIENTCOUNT>
```

Use this flag to specify the current *CLIENTCOUNT*. Default is None. Remember that a number >=25 is required to enable activation of client OSes while for server OSes and Office >=5.

```
-a or --activation-interval <ACTIVATIONINTERVAL>
```

Instructs clients to retry activation every *ACTIVATIONINTERVAL* minutes if it was unsuccessful, e.g. because it could not reach the server. The default is 120 minutes (2 hours).

```
-r or --renewal-interval <RENEWALINTERVAL>
```

Instructs clients to renew activation every *RENEWALINTERVAL* minutes. The default is 10080 minutes (7 days).

```
-s or --sqlite [<SQLFILE>]
```

Use this option to store request information from unique clients in an SQLite database. Deactivated by default.

```
-t0 or --timeout-idle <TIMEOUTIDLE>
```

Maximum inactivity time (in seconds) after which the connection with the client is closed. Default setting is serve forever (no timeout).

```
-t1 or --timeout-sndrcv <TIMEOUTSNDRCV>
```

Set the maximum time (in seconds) to wait for sending / receiving a request / response. Default is no timeout.

```
-y or --async-msg
```

With high levels of logging (e.g hundreds of log statements), in a traditional synchronous log model, the overhead involved becomes more expensive, so using this option you enable printing (pretty / logging) messages asynchronously reducing time-consuming. Deactivated by default.

```
-V or --loglevel <{CRITICAL, ERROR, WARNING, INFO, DEBUG, MININFO}>
```

Use this flag to set a logging loglevel. The default is *ERROR*. example:

```
user@host ~/path/to/folder/py-kms $ python3 pykms_Server.py -V INFO
```

creates *pykms_logserver.log* with these initial messages:

```
Mon, 12 Jun 2017 22:09:00 INFO     TCP server listening at :: on port 1688.
Mon, 12 Jun 2017 22:09:00 INFO     HWID: 364F463A8863D35F
```

```
-F or --logfile <LOGFILE>
```

Creates a *LOGFILE.log* logging file. The default is named *pykms_logserver.log*. example:

```
user@host ~/path/to/folder/py-kms $ python3 pykms_Server.py 192.168.1.102 1688 -F ~/path/
↪to/folder/py-kms/newlogfile.log -V INFO -w RANDOM
```

creates *newlogfile.log* with these initial messages:

```
Mon, 12 Jun 2017 22:09:00 INFO     TCP server listening at 192.168.1.102 on port 1688.
Mon, 12 Jun 2017 22:09:00 INFO     HWID: 58C4F4E53AE14224
```

You can also enable other suboptions of `-F` doing what is reported in the following table:

```
-S or --logsize <MAXSIZE>
```

Use this flag to set a maximum size (in MB) to the output log file. Deactivated by default.

subparser `connect`

```
-n or --listen <'IP,PORT'>
```

Use this option to add multiple listening ip address - port couples. Note the format with the comma between the ip address and the port number. You can use this option more than once.

```
-b or --backlog <BACKLOG>
```

Use this option to specify the maximum length of the queue of pending connections, referred to a ip address - port couple. If placed just after `connect` refers to the main address and all additive couples without `-b` option. Default is 5.

```
-u or --no-reuse
```

Use this option not to allow binding / listening to the same ip address - port couple specified with `-n`. If placed just after `connect` refers to the main address and all additive couples without `-u` option. Reusing port is activated by default (except when running inside the Windows Sandbox and the current user is `WDAGUtilityAccount`).

```
-d or --dual
```

Use this option to allow listening to an IPv6 address also accepting connections via IPv4. If used it refers to all addresses (main and additional). Deactivated by default.

examples (with fictitious addresses and ports):

6.1.2 pykms_Client.py

If *py-kms* server doesn't works correctly, you can test it with the KMS client `pykms_Client.py`, running on the same machine where you started `pykms_Server.py`.

For example (in separated bash windows) run these commands:

```
user@host ~/path/to/folder/py-kms $ python3 pykms_Server.py -V DEBUG
user@host ~/path/to/folder/py-kms $ python3 pykms_Client.py -V DEBUG
```

If you wish to get KMS server from DNS server: (ie perform a DNS resolution on _vlmcs._tcp.domain.tld, if ever there are several answers, only the first one is selected.). Althought that mode is supposed to be specific to devices connect

to an Active Directory domain, setting a fully qualified name and a workgroup may help to use that automatic KMS discovery feature.

```
user@host ~/path/to/folder/py-kms $ python3 pykms_Client.py -V DEBUG -F STDOUT -D␣
↪contoso.com
user@host ~/path/to/folder/py-kms $ python3 pykms_Client.py -V DEBUG -F STDOUT -D␣
↪contoso.com
```

Or if you want better specify:

```
user@host ~/path/to/folder/py-kms $ python3 pykms_Server.py <YOUR_IPADDRESS> 1688 -V␣
↪DEBUG
user@host ~/path/to/folder/py-kms $ python3 pykms_Client.py <YOUR_IPADDRESS> 1688 -V␣
↪DEBUG
```

You can also put further parameters as defined below:

```
ip <IPADDRESS>
```

Define *IPADDRESS* (or hostname) of py-kms' KMS Server. This parameter is always required.

```
port <PORT>
```

Define TCP *PORT* the KMS service is listening on. Default is 1688.

```
-m or --mode <{WindowsVista, Windows7, Windows8, Windows8.1, Windows10, Office2010,␣
↪Office2013, Office2016, Office2019}>
```

Use this flag to manually specify a Microsoft *PRODUCTNAME* for testing the KMS server. Default is Windows8.1.

```
-c or --cmid <CMID>
```

Use this flag to manually specify a CMID to use. If no CMID is specified, a random one will be generated. The Microsoft KMS host machine identifies KMS clients with a unique Client Machine ID (CMID, example: ae3a27d1-b73a-4734-9878-70c949815218). For a KMS client to successfully activate, the KMS server needs to meet a threshold, which is a minimum count for KMS clients. Once a KMS server records a count which meets or exceeds threshold, KMS clients will begin to activate successfully. Each unique CMID recorded by KMS server adds towards the count threshold for KMS clients. This are retained by the KMS server for a maximum of 30 days after the last activation request with that CMID. Note that duplicate CMID only impacts on KMS server machine count of client machines. Once KMS server meets minimum threshold, KMS clients will activate regardless of CMID being unique for a subset of specific machines or not.

```
-n or --name <MACHINENAME>
```

Use this flag to manually specify an ASCII *MACHINENAME* to use. If no *MACHINENAME* is specified a random one will be generated.

```
-t0 or --timeout-idle <TIMEOUTIDLE>
```

Set the maximum time (in seconds) to wait for a connection attempt to KMS server to succeed. Default is no timeout.

```
-t1 or --timeout-sndrcv <TIMEOUTSNDRCV>
```

Set the maximum time (in seconds) to wait for sending / receiving a request / response. Default is no timeout.

```
-y or --async-msg
```

Prints pretty / logging messages asynchronously. Deactivated by default.

```
-V or --loglevel <{CRITICAL, ERROR, WARNING, INFO, DEBUG, MININFO}>
```

Use this flag to set a logging loglevel. The default is *ERROR*.

```
-F or --logfile <LOGFILE>
```

Creates a *LOGFILE.log* logging file. The default is named *pykms_logclient.log*. You can enable same *pykms_Server.py* suboptions of `-F`.

```
-S or --logsize <MAXSIZE>
```

Use this flag to set a maximum size (in MB) to the output log file. Deactivated by default.

6.2 Docker Environment

This are the currently used `ENV` statements from the Dockerfile(s). For further references what exactly the parameters mean, please see the start parameters for the *server*.

```
# IP-address
# The IP address to listen on. The default is "::" (all interfaces).
ENV IP ::

# TCP-port
# The network port to listen on. The default is "1688".
ENV PORT 1688

# ePID
# Use this flag to manually specify an ePID to use. If no ePID is specified, a random␣
↪ePID will be generated.
ENV EPID ""

# lcid
# Use this flag to manually specify an LCID for use with randomly generated ePIDs.␣
↪Default is 1033 (en-us).
ENV LCID 1033

# The current client count
# Use this flag to specify the current client count. Default is 26.
# A number >=25 is required to enable activation of client OSes; for server OSes and␣
↪Office >=5.
ENV CLIENT_COUNT 26

# The activation interval (in minutes)
# Use this flag to specify the activation interval (in minutes). Default is 120 minutes␣
↪(2 hours).
ENV ACTIVATION_INTERVAL 120
```

(continues on next page)

(continued from previous page)

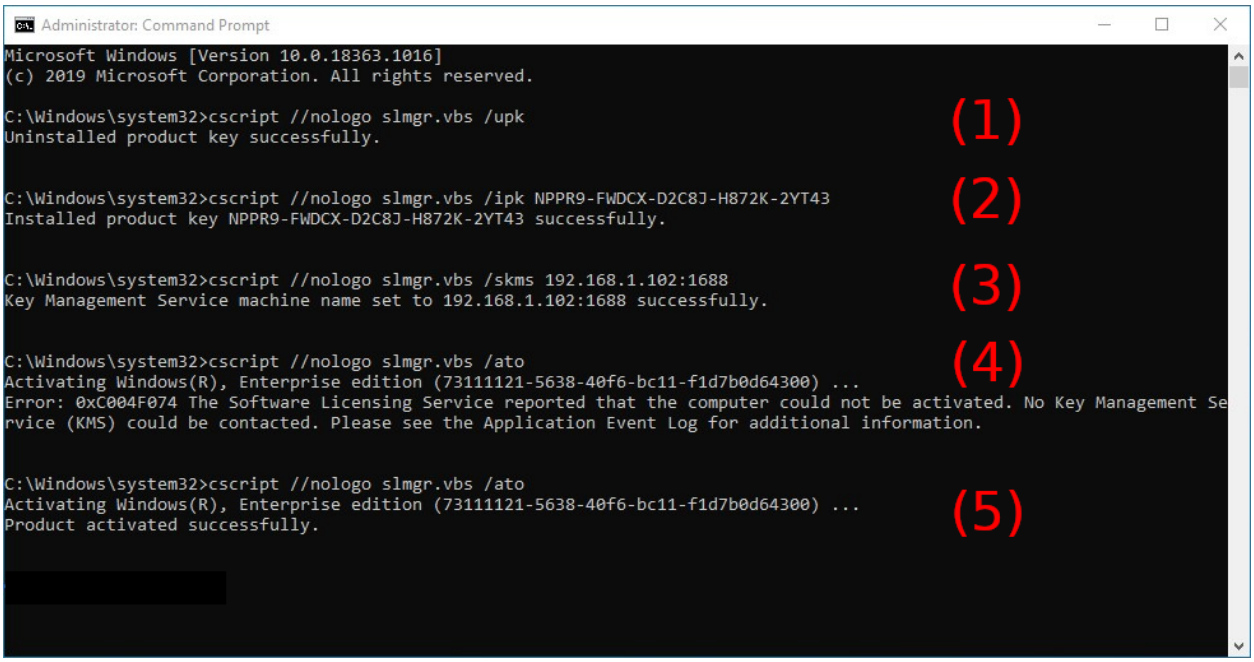
```
# The renewal interval (in minutes)
# Use this flag to specify the renewal interval (in minutes). Default is 10080 minutes␣
↪(7 days).
ENV RENEWAL_INTERVAL 10080

# hwid
# Use this flag to specify a HWID.
# The HWID must be an 16-character string of hex characters.
# The default is "RANDOM" to auto-generate the HWID or type a specific value.
ENV HWID RANDOM

# log level ("CRITICAL", "ERROR", "WARNING", "INFO", "DEBUG")
# Use this flag to set a Loglevel. The default is "ERROR".
ENV LOGLEVEL ERROR

# Log file
# Use this flag to set an output Logfile. The default is "/var/log/pykms_logserver.log".
ENV LOGFILE /var/log/pykms_logserver.log

# Log file size in MB
# Use this flag to set a maximum size (in MB) to the output log file. Deactivated by␣
↪default.
ENV LOGSIZE ""
```

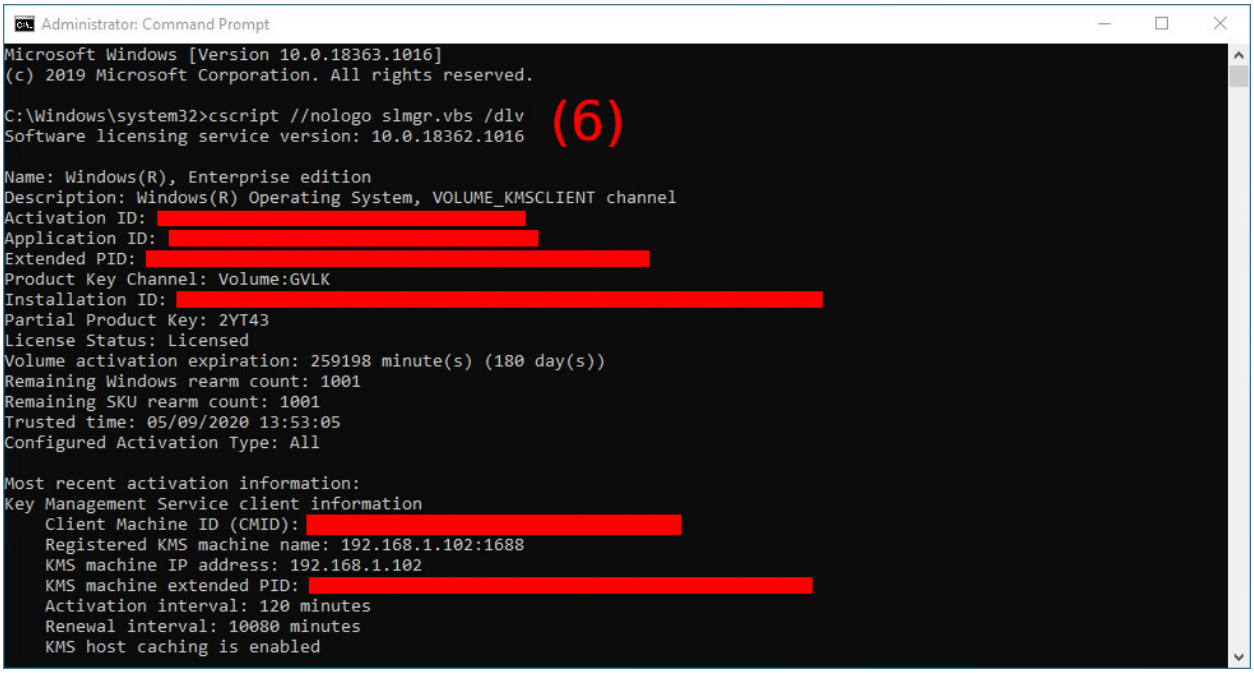
6.3 Activation Procedure

The product asks for a key during installation, so it needs you to enter the GVLK. Then the user can set connection parameters, while KMS server must already be running on server machine. Finally with specific commands, activation occurs automatically and can be extended later every time for another 180 (or 30 or 45) days.

6.3.1 Windows

The `//nologo` option of `cscript` was used only to hide the startup logo.





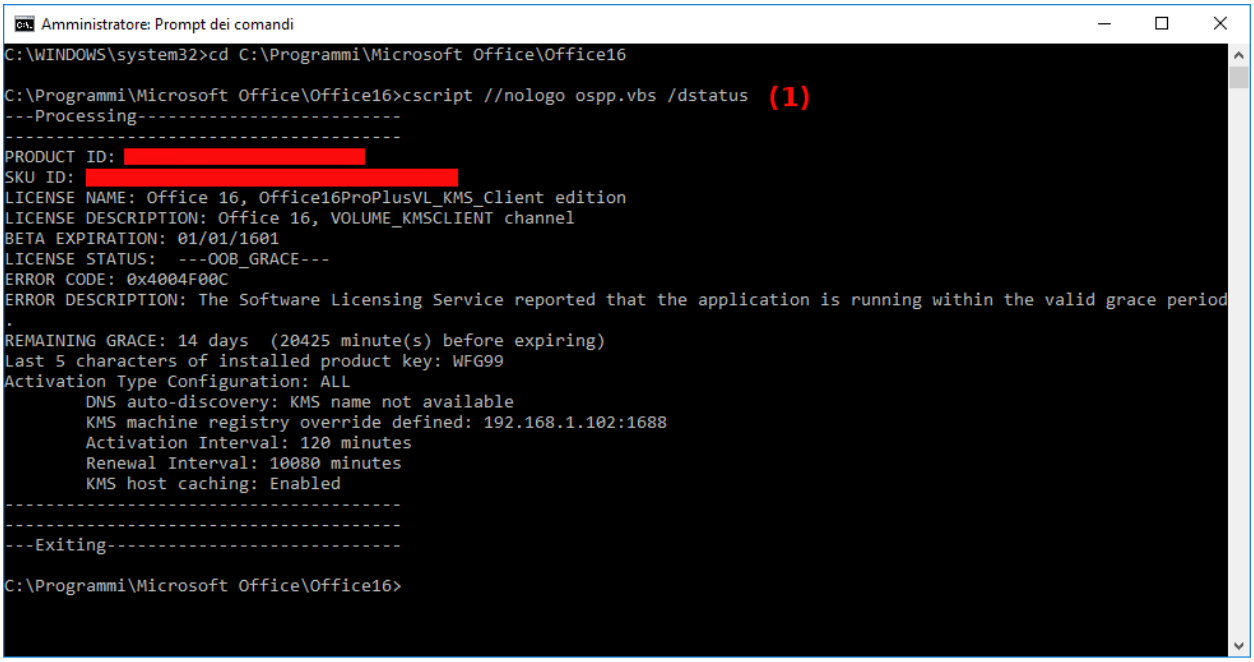
0. Run a Command Prompt as Administrator (you are directly in `C:\Windows\System32` path).
1. This is optional, it's for unistalling any existing product key.
2. Then put in your product's GVLK.
3. Set connection parameters.
4. Try online activation, but if that fails with error...
   - `0xC004F074` You'll most likely have to configure your firewall that it accepts incoming connections on TCP port 1688. So for Linux users (server-side with `pykms_Server.py` running): `sudo ufw allow 1688` (revert this rule `sudo ufw delete allow 1688`) should fix that.

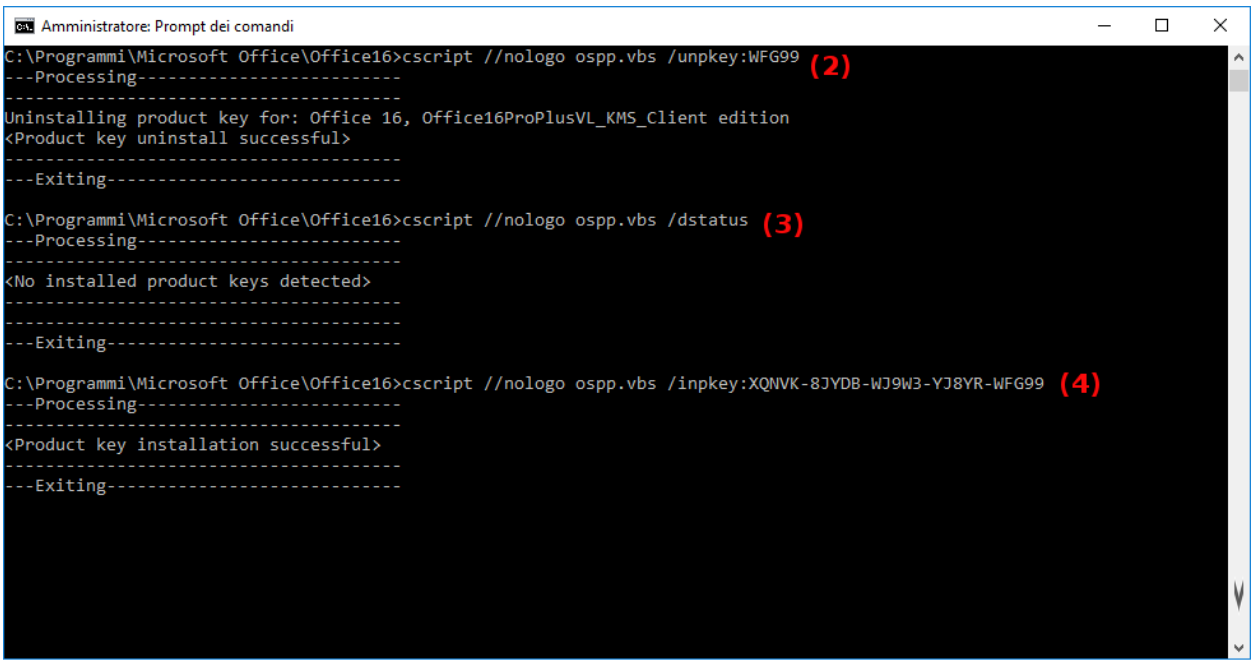
- `0xC004F069` Take a look into the issue here, it will may help you…

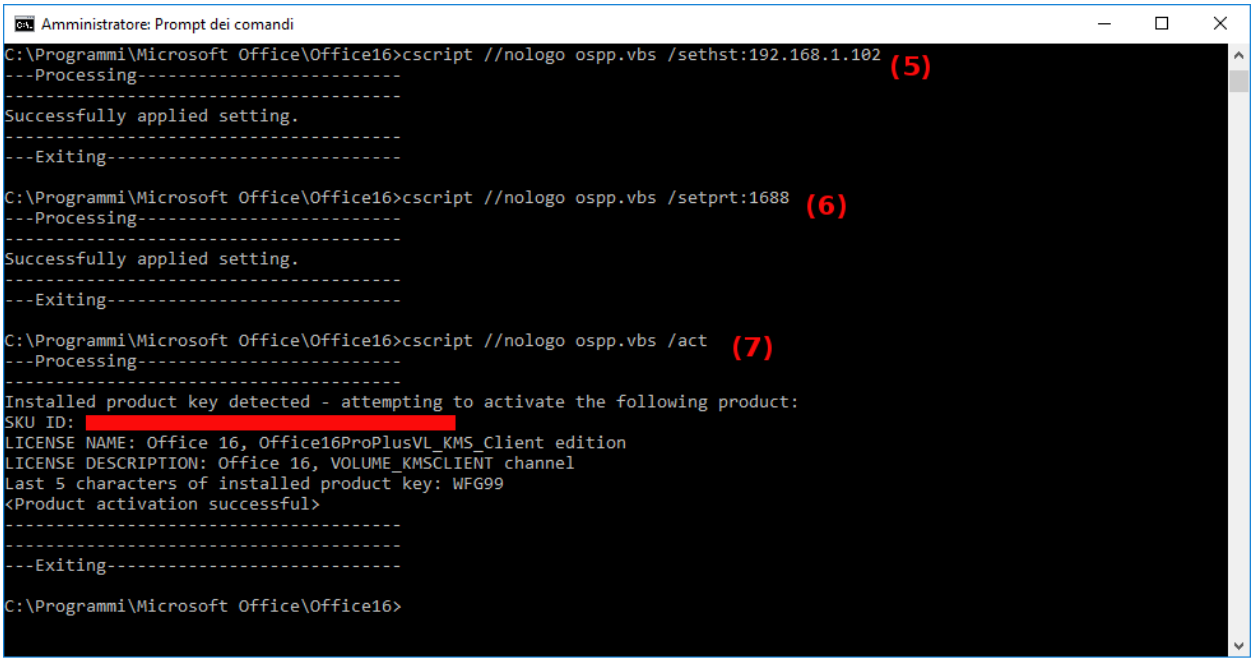
5. Attempt online activation (now with traffic on 1688 enabled).
6. View license informations (optional).

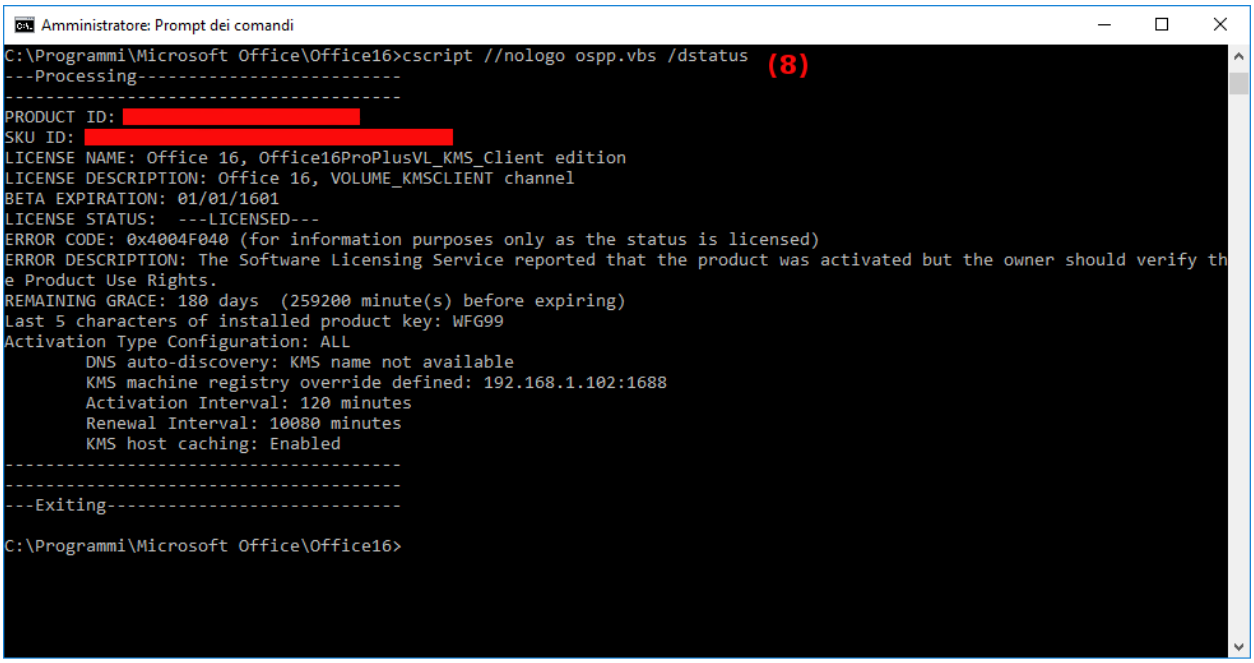
6.3.2 Office

Note that you'll have to install a volume license (VL) version of Office. Office versions downloaded from MSDN and / or Technet are non-VL.









0. Run a Command Prompt as Administrator and navigate to Office folder `cd C:\ProgramFiles\Microsoft Office\OfficeXX` (64-bit path) or `cd C:\ProgramFiles(x86)\Microsoft Office\OfficeXX` (32-bit path), where XX = 14 for Office 2010, 15 for Office 2013, 16 for Office 2016 or Office 2019.
1. As you can see, running `/dstatus`, my Office is expiring (14 days remaining).
2. Only for example, let's go to uninstall this product.
3. This is confirmed running `/dstatus` again.
4. Now i put my product's GVLK (and you your key).
5. Set the connection parameter KMS server address.
6. Set the connection parameter KMS server port.

7. Activate installed Office product key.
8. View license informations (in my case product is now licensed and remaining grace 180 days as expected).

CHAPTER

SEVEN

CHANGELOG

7.1 py-kms_2022-12-16

- Added support for new web-gui into Docker
- Implemented whole-new web-based GUI with Flask
- Removed old GUI (Etrigan) from code and resources
- Removed sqliteweb
- Removed Etrigan (GUI)

7.2 py-kms_2022-12-07

- Added warning about Etrigan (GUI) being deprecated
- More docs (do not run on same machine as client)
- Added Docker support for multiple listen IPs
- Added graceful Docker shutdowns

7.3 py-kms_2021-12-23

- More Windows 10/11 keys
- Fixed some deprecation warnings
- Fixed SO_REUSEPORT platform checks
- Fixed loglevel “MININFO” with Docker
- Added Docker healthcheck
- Added UID/GID change support for Docker
- Dependabot alerts

7.4 py-kms_2021-10-22

- Integrated Office 2021 GLVK keys & database
- Docker entrypoint fixes
- Updated docs to include SQLite stuff
- Fix for undefined timezones
- Removed LOGFILE extension checks
- Added support for Windows 11

7.5 py-kms_2021-10-07

- Helm charts for Kubernetes deployment
- Windows 2022 updates
- Faster Github Action builds

7.6 py-kms_2021-11-12

- Addded GHCR support
- Docs table reformatted
- Updated GUI
- Windows Sandbox fix
- Added contribution guidelines
- Docker multiarch
- Reshot screenshots in docs

7.7 py-kms_2020-10-01

- Sql database path customizable.
- Sql database file keeps different AppId.
- Support for multi-address connection.
- Added timeout send / receive.

7.8 py-kms_2020-07-01

- py-kms Gui: now matches all cli options, added modes onlyserver / onlyclient, added some animations.
- Added suboptions FILEOFF and STDOUTOFF of -F.
- Created option for asynchronous messages.
- Cleaned options parsing process.

7.9 py-kms_2020-02-02

- Optimized pretty-print messages process.
- Added -F FILESTDOUT option.
- Added deamonization options (via Etrigan project).
- py-kms GUI resurrected (and improved).
- Cleaned, cleaned, cleaned.

7.10 py-kms_2019-05-15

- Merging for Python2 / Python3 compatibility all-in-one.
- Added new options:
  - timeout, logsize.
  - more control on logging and info visualization (custom loglevel and stdout logfile redirection) to match this request.
- Setup for multithreading support.
- Graphical improvements:
  - added a (*"really silly"*) tkinter GUI as an alternative to command line.
- Dockerized with sqlite-web.
- Fixed activation threshold.
- Renamed files, cosmetics and many other little big adjustments.

7.11 py-kms_2018-11-15

- Implemented some good modifications inspired by this other fork.
  - Clean up code (deleted no longer useful files randomHWID.py, randomEPID.py, timezones.py; erased useless functions and import modules)
  - Reading parameters directly from a slightly modified KmsDataBase.xml (created with LicenseManager 5.0 by Hotbird64 HGM) with kmsDB2Dict.py
- Added support for Windows Server 2019 and Office 2019.
- Improved random EPID generation.

- Corrected this in kmsBase.py

7.12 py-kms_2018-03-01

- *py-kms NOW is for Python3 too (py3-kms), the previous one (written with Python2) is renamed py2-kms*
- *Repaired logging messages*
- *Added pretty processing messages*

7.13 py-kms_2017-06-01

- *Added option verbose logging in a file*
- *Updated "kmsBase.py" with new SKUIDs*
- *Added a brief guide "py-kms-Guide.pdf" (replaced "client-activation.txt")*
- *Added a well formatted and more complete list of volume keys "py-kms-ClientKeys.pdf" (replaced "client-keys.txt")*

7.14 py-kms_2016-12-30

- *Updated kmsBase.py (Matches LicenseManager 4.6.0 by Hotbird64 HGM)*

7.15 py-kms_2016-08-13

- *Fixed major bug on Response function*
- *Fixed Random PID Generator (thanks: mkuba50)*

7.16 py-kms_2016-08-12

- *Added missing UUID, credits: Hotbird64*
- *Added Windows Server 2016 in random PID generator*

7.17 py-kms_2016-08-11

- *Added Windows Server 2016 UUID*
- *Fixed GroupID and PIDRange*
- *Added Office 2016 CountKMSID*

7.18 py-kms_2015-07-29

- *Added Windows 10 UUID*

7.19 py-kms_2014-10-13 build 3:

- *Added Client Activation Examples: "client-activation.txt"*
- *Added Volume Keys: "client-keys.txt"*

7.20 py-kms_2014-10-13 build 2:

- *Added missing skuIds in file "kmsbase.py". Thanks (user_hidden)*

7.21 py-kms_2014-10-13 build 1:

- *The server now outputs the hwid in use.*
- *The server hwid can be random by using parameter: "-w random". Example: "python server.py -w random"*
- *Included file "randomHWID.py" to generate random hwid on demand.*
- *Included file "randomPID.py" to generate random epid and hwid on demand.*

7.22 py-kms_2014-03-21T232943Z:

- *The server HWID can now be specified on the command line.*
- *The client will print the HWID when using the v6 protocol.*

7.23 py-kms_2014-01-03T032458Z:

- *Made the sqlite3 module optional.*
- *Changed the "log" flag to an "sqlite" flag and made a real log flag in preparation for when real request logging is implemented.*

7.24 py-kms_2014-01-03T025524Z:

- *Added RPC response decoding to the KMS client emulator.*

7.25 py-kms_2013-12-30T064443Z:

- *The v4 hash now uses the proper pre-expanded key.*

7.26 py-kms_2013-12-28T073506Z:

- *Modified the v4 code to use the custom aes module in order to make it more streamlined and efficient.*

7.27 py-kms_2013-12-20T051257Z:

- *Removed the need for the pre-computed table (tablecomplex.py) for v4 CMAC calculation, cutting the zip file size in half.*

7.28 py-kms_2013-12-16T214638Z:

- *Switched to getting the to-be-logged request time from the KMS server instead of the client.*

7.29 py-kms_2013-12-16T030001Z:

- *You can now specify the CMID and the Machine Name to use with the client emulator.*

7.30 py-kms_2013-12-16T021215Z:

- *Added a request-logging feature to the server. It stores requests in an SQLite database and uses the ePIDs stored there on a per-CMID basis.*
- *The client emulator now works for v4, v5, and v6 requests.*
- *The client emulator now also verifies the KMS v4 responses it receives.*

7.31 py-kms_2013-12-14T230215Z

- *Added a client (work in progress) that right now can only generate and send RPC bind requests.*
- *Added a bunch of new classes to handle RPC client stuff, but I might just end up moving their functions back into the old classes at some point.*
- *Lots of other code shuffling.*
- *Made the verbose and debug option help easier to read.*
- *Added some server error messages.*

7.32 py-kms_2013-12-08T051332Z:

- *Made some really huge internal changes to streamline packet parsing.*

7.33 py-kms_2013-12-06T034100Z:

- *Added tons of new SKU IDs*

7.34 py-kms_2013-12-05T044849Z:

- *Added Office SKU IDs*
- *Small internal changes*

7.35 py-kms_2013-12-04T010942Z:

- *Made the rpcResponseArray in rpcRequest output closer to spec*

7.36 py-kms_2013-12-01T063938Z:

- *SKUID conversion: Converts the SKUID UUID into a human-readable product version for SKUIDs in its SKUID dictionary.*
- *Fancy new timezone conversion stuff.*
- *Enabled setting custom LCIDs.*
- *Data parsing is now handled by structure.py.*
- *Some other minor stuff you probably won't notice.*

7.37 py-kms_2013-11-27T061658Z:

- *Got rid of custom functions module (finally)*

7.38 py-kms_2013-11-27T054744Z:

- *Simplified custom functions module*
- *Got rid of "v4" subfolder*
- *Cleaned up a bunch of code*

7.39 py-kms_2013-11-23T044244Z:

- *Added timestamps to verbose output*
- *Made the verbose output look better*

7.40 py-kms_2013-11-21T014002Z:

- *Moved some stuff into verbose output*
- *Enabled server ePIDs of arbitrary length*

7.41 py-kms_2013-11-20T180347Z:

- *Permanently fixed machineName decoding*
- *Adheres closer to the DCE/RPC protocol spec*
- *Added client info to program output*
- *Small formatting changes*

7.42 py-kms_2013-11-13:

- *First working release added to the Mega folder.*

CHAPTER

EIGHT

README

Keep in mind that this project is not intended for production use. Feel free to use it to test your own systems or maybe even learn something from the protocol structure. :)

8.1 History

py-kms is a port of node-kms created by cyrozap, which is a port of either the C#, C++, or .NET implementations of KMS Emulator. The original version was written by CODYQX4 and is derived from the reverse-engineered code of Microsoft's official KMS. This version of *py-kms* is for itself a fork of the original implementation by SystemRage, which was abandoned early 2021.

8.2 Features

- Responds to `v4`, `v5`, and `v6` KMS requests.
- Supports activating:
  - Windows Vista
  - Windows 7
  - Windows 8
  - Windows 8.1
  - Windows 10 (1511 / 1607 / 1703 / 1709 / 1803 / 1809)
  - Windows 10 (1903 / 1909 / 20H1, 20H2, 21H1, 21H2)
  - Windows 11 (21H2)
  - Windows Server 2008
  - Windows Server 2008 R2
  - Windows Server 2012
  - Windows Server 2012 R2
  - Windows Server 2016
  - Windows Server 2019

- Windows Server 2022
- Microsoft Office 2010 (Volume License)
- Microsoft Office 2013 (Volume License)
- Microsoft Office 2016 (Volume License)
- Microsoft Office 2019 (Volume License)
- Microsoft Office 2021 (Volume License)
- It's written in Python (tested with Python 3.10.1).
- Supports execution by `Docker`, `systemd` and many more...
- Uses `sqlite` for persistent data storage (with a simple web-based explorer).

8.3 Documentation

The wiki has been completly reworked and is now available on readthedocs.io. It should provide you all the necessary information about how to setup and to use *py-kms* , all without clumping this readme. The documentation also houses more details about activation with *py-kms* and how to get GVLK keys.

8.4 Quick start

- To start the server, execute `python3 pykms_Server.py [IPADDRESS] [PORT]`, the default *IPADDRESS* is `::` (all interfaces) and the default *PORT* is `1688`. Note that both the address and port are optional. It's allowed to use IPv4 and IPv6 addresses. If you have a IPv6-capable dual-stack OS, a dual-stack socket is created when using a IPv6 address. **In case your OS does not support IPv6, make sure to explicitly specify the legacy IPv4 of `0.0.0.0`!**
- To start the server automatically using Docker, execute `docker run -d --name py-kms --restart always -p 1688:1688 ghcr.io/py-kms-organization/py-kms`.
- To show the help pages type: `python3 pykms_Server.py -h` and `python3 pykms_Client.py -h`.

8.5 License

- *py-kms* is