**Private Pre-Shared Key Sync with Azure Active Directory Guide**

by Tim Smith, SA – 06/21/2022 – v1.0.0

Overview**:**

This guide covers how to set up and run the script to sync your Azure (cloud based) Active directory users with the Private Pre-shared-key (PPSK) with in ExtremeCloud IQ (XIQ) public cloud only. PPSK is a solution provided by Extreme Networks to fill in the gap between a Wi-Fi SSID solution using a single PSK for all users and deploying a full 802.1X solution. Extreme Networks PPSK solution allows the ability to create a dedicated key for each individual user or device on the same SSID, providing the ability to lower the number of SSIDs broadcasting in the air, minimizing the airtime consumption due to overhead management frames. This solution also adds the ability to assign VLANs based on user/device groups to avoid the need of separate SSIDs to segregate these groups.

This guide enables you to leverage your existing Azure Active Directory security groups to create automatically a Private Pre-shared key for every AD user, remove the PPSK user if a user is disabled or removed from the group in Azure AD.

For this script to work properly, each AD user will need to have a unique email address.

Target Audience**:** Technical

# 

# PPSK Use Cases:

* Identity for IoT devices
* BYOD for employees
* Staff device onboarding
* Secure Guest Onboarding (time-based keys with employee sponsorship)
* Hospitality vertical using the hyper-segmentation feature, Private Client Groups (PCGs)
* Third-party via API integration

Table of Contents

[Prerequisites: 4](#_Toc106803597)

[Scripting Environment Preparation: 4](#_Toc106803598)

[Information: 4](#_Toc106803599)

[Device Choice: 4](#_Toc106803600)

[Python Installation: 4](#_Toc106803601)

[4](#_Toc106803602)

[Mac OSX Big Sur 5](#_Toc106803603)

[Windows 10 5](#_Toc106803604)

[Required Modules: 5](#_Toc106803605)

[Checking for existing Modules 5](#_Toc106803606)

[Installing required modules 5](#_Toc106803607)

[5](#_Toc106803608)

[Script Variables: 6](#_Toc106803609)

[PCG Support (optional) 6](#_Toc106803610)

[Generating the XIQ Token 7](#_Toc106803611)

[Swagger 7](#_Toc106803612)

[Login 7](#_Toc106803613)

[Authorize in Swagger 8](#_Toc106803614)

[Generating Specific Tokens 9](#_Toc106803615)

[Creating the Azure AD App 10](#_Toc106803616)

[Granting Permissions to the app 10](#_Toc106803617)

[Creating Client Secret 11](#_Toc106803618)

[AD Group Object ID 12](#_Toc106803619)

[XIQ User Group ID 12](#_Toc106803620)

[XIQ Network Policy ID 13](#_Toc106803621)

[Running the Script: 14](#_Toc106803622)

[Log File 14](#_Toc106803623)

[Scheduling Script to Run 14](#_Toc106803624)

[Mac & Linux based Systems 14](#_Toc106803625)

[Setting up a Cron Job 14](#_Toc106803626)

[Cron Job Time Format 15](#_Toc106803627)

[Cron Job Script and Script Location 15](#_Toc106803628)

[Cron Job Output and Job Completion 15](#_Toc106803629)

[Cron Job Command Example 16](#_Toc106803630)

[Windows based Systems 16](#_Toc106803631)

[Setting up Windows Task Scheduler 16](#_Toc106803632)

[Start a Program 16](#_Toc106803633)

[Editing the Time 17](#_Toc106803634)

[Troubleshooting: 18](#_Toc106803635)

[Log File 18](#_Toc106803636)

[Invalid XIQ token 18](#_Toc106803637)

[Invalid XIQ token format 18](#_Toc106803638)

[Expired XIQ token – Code 401 & JWT expired 18](#_Toc106803639)

[Invalid XIQ Username/password – Code 401 18](#_Toc106803640)

[Failed Authentication for Azure AD API 18](#_Toc106803641)

[Invalid or Wrong Group ID entered 18](#_Toc106803642)

[XIQ User Failed to Create – Code 400 19](#_Toc106803643)

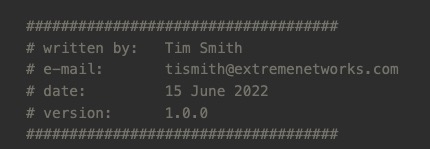
[XIQ Timeout Error 19](#_Toc106803644)

[Email is not set for AD User 19](#_Toc106803645)

[Intermittent API issue 19](#_Toc106803646)

[Azure\_AD\_Test.py 19](#_Toc106803647)

# Prerequisites:

* ExtremeCloud IQ Public Cloud, Private Cloud (IQVA on-prem is not supported.)  
  o The key directory can be stored in the cloud (unlimited keys) or local on all access points (10,000 key maximum limit)
* Knowledge of XIQ by adding access points, creating network policies, and SSIDs
* XIQ PPSK SSID and associated User Groups configured
* RadSec Proxy requires TCP Port 2083 to be open on your internet firewall
* One or more XIQ native access points
* Not supported on wired systems, A3 NAC, or campus-based Wi-Fi systems (WiNG or IdentiFi)
* Download of the following files
  + XIQ-Azure-AD-PPSK-Sync.py
    - Version 1.0.0 is the current version. See lines 8-13 in script
  + Azure\_AD\_Test.py (optional – see [troubleshooting](#_Troubleshooting:) section)
    - Version 1.0.0 is the current version. Lines 6-11
  + requirements.txt (optional – see [modules](#_Installing_required_modules) section)

# 

# Scripting Environment Preparation:

### Information:

The XIQ-Azure-AD-PPSK-Sync.py script requires, at minimum, Python 3.6 and tested up to Python 3.9. This script can be executed manually, but ideally would be set up as a cronjob to be ran ever 8, 12, or 24 hours. This script can be executed from any device with python and the needed modules installed. This device will need to be able to reach the Azure Cloud as well as reach out to ExtremeCloud IQ.

The script, when ran, will create a XIQ-Azure-AD-PPSK-Sync.log file. This log file will show information about PPSK users created and deleted. It will also show how many users were parsed from XIQ and Active Directory when ran. Any API errors experienced will also show up in the log file.

## 

## Device Choice:

This script can be executed from any device that can run python 3.6 or higher. The device could be a server running Redhat, a pc/laptop running Windows 10 or Mac OSX, or even a Raspberry Pi type device. The device will need to be on the network and be able to reach the local Active Directory server as well as reach ExtremeCloud IQ. This can be done through a proxy. Proxy config is beyond the scope of this guide.

## 

## Python Installation:

Depending on the device that is used, you may need to install python, or a different version of python. The easiest way to check the version of python is to open the terminal (Power Shell on Windows) and type this command.

### 

python3 --version

Below are some examples of installing python3 for Windows and Mac OSX. Linux systems that were tested all had python3.6 or higher installed by default

### Mac OSX Big Sur

* Open the terminal and enter python3 –version
  + This triggers the install of Developer Tools
* Graphical user interface, text

  Description automatically generatedClick Install
* Click Agree
* pip3 is needed to install python modules
* With Big Sur the Developer tools does not install pip3
* Mac terminal will be used to install pip3
* Running this command will check if pip is installed

pip3 --version

* Run the following command to install

curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py && python3 get-pip.py

### Windows 10

* Search Microsoft store for python 3.9 and click install
* Graphical user interface, text, application, Teams

  Description automatically generatedLog in with Microsoft credentials
* The Windows store installs pip3 with python3. Pip3 will be used to install the needed modules

## Required Modules:

The **requests** module is the only module required for the XIQ-Azure-AD-PPSK-Sync.py script. For Linux and Mac

### Checking for existing Modules

You can check to see if the required module is installed using the terminal (PowerShell for Windows). Run the following command.

python3 -c “import requests”

The module is not installed if a ‘*ModuleNotFoundError: No module named '<module name>*' error is returned.

### Installing required modules

The required module can be installed using pip3 using the downloaded requirements.txt file with the following command.

## 

pip3 install -r requirements.txt

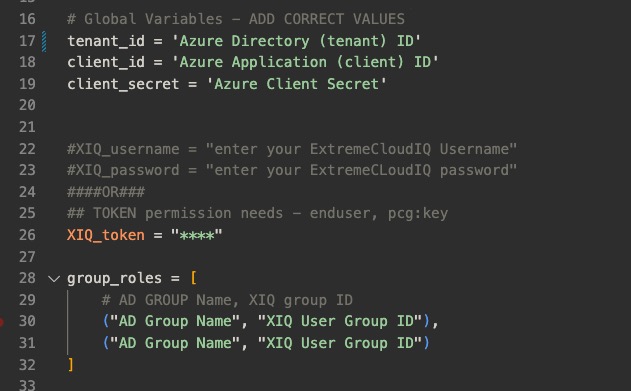
Or the module can be installed individually using

pip3 install requests

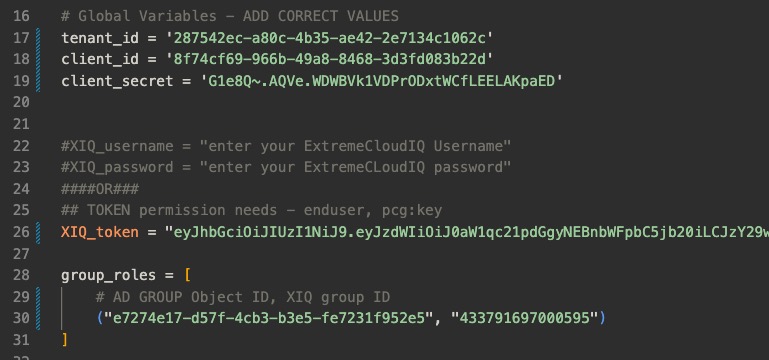
# Script Variables:

The Global Variable section of the script (file name: XIQ-Azure-AD-PPSK-Sync.py) will need to be updated with the correct values. We will briefly cover each of these and for some will go into more detail below.

**Lines 17 – 19** are info for Azure Active Directory

1. **tenant\_id** – Azure Directory (tenant) ID created when creating the Azure App. See [creating the Azure app](#_Granting_Permissions_to).
2. **client\_id** – Azure Application (client) ID created when creating the Azure App. See [creating the Azure app](#_Granting_Permissions_to).
3. **client\_secret -** Azure Client Secret created when creating the Azure App. See [creating the Azure app](#_Creating_Client_Secret).

**Lines 22 – 26** are for XIQ authentication. There are 2 methods that could be used, but the token method is recommended.

1. XIQ username and password –
   * **Lines 22 and 23** - uncomment by 16deleted the **#** at the beginning of the line. Then fill in the username and password
   * **Line 26** - comment out line by adding # to the beginning of the line.
2. Token method – a token can be generated to only allow access to view/create/delete PPSK users. This is the preferred method. Details on how to generate this token is below in the [Generating the XIQ Token](#_Generating_the_XIQ_1) and specifically the [Generating Specific Tokens](#_Generating_Specific_Tokens) sub section.

**Lines 28 – 32** are where we will define which AD groups to sync with which XIQ User Groups. Note that the brackets around the groups create a list. Each set of groups should be in a set of parentheses and each element separated by a coma. Each set of groups should be separated by a coma. If only one set of groups is required, **line 31** can be deleted and the coma at the end of **line 30** removed.

* We will cover how to get the needed AD Group Object in the [AD Group Object ID](#_AD_Group_Object) section
* We will cover how to get the XIQ User Group ID in the [XIQ User Group ID](#_XIQ_User_Group) section
* **NOTE**: The order is very important here. If the same AD user is in multiple groups, the user will be put in the first XIQ User Group in the list. XIQ users can only be in one PPSK User group.

#### PCG Support (optional)

**Line 34** To enable PCG Support change the **PCG\_Enable** Variable from **False** to **True**

**Lines 36-42** If PCG is Enabled **PCG\_Maping** should be updated with correct information. If PCG is not Enabled **PCG\_Maping** will not be used and does not need to be updated.

1. **Line 37** – This should be replaced with the XIQ User Group ID number that correlates with the PCG. See [XIQ User Group ID](#_XIQ_User_Group) section
2. **Line 38** – This is the name of the User Group associated with the ID on line 42
   * This is needed to be able to add and remove Users from the PCG.
3. **Line 39** – This is the Network Policy ID associated with the PCG
   * We will cover how to get the Network Policy ID in the [Network Policy ID](#_XIQ_Network_Policy) section
4. **Line 40** – This is the Network Policy Name associated with the PCG

## Generating the XIQ Token

You can view our developer portal site at <https://developer.extremecloudiq.com/> There is a link to our swagger page here, as well as other developer tools. There is also a Communities section to reach out with any questions.

#### Swagger

We will use the swagger interface to generate the token <https://api.extremecloudiq.com/>  
In the swagger page, clicking on any API will expand out information about the API as well as give you the ability to try it. Clicking the “Try it out” button, fill out any needed information, and then click the execute button will allow you to try that specific API call.

The 2nd generation APIs are based on access tokens that are generated by an XIQ account. External XIQ accounts need to be granted API to be able to leverage the new APIs. See [A Guide to Getting Started with v2 APIs in XIQ](https://extremeportal.force.com/ExtrArticleDetail?an=000102173) for more details. Currently these tokens can only be generated through the */login* POST API request. They cannot be generated through the XIQ GUI.

Graphical user interface

Description automatically generated with medium confidence

##### Login

*Request Body*

{  
 "username": "xiq@example.com",  
 "password": "changeme"  
}

The */login* POST request is used to generate an access token. In the request body enter a local administrator XIQ account username and password and the API will respond with an access token that can be used for any following calls. This token will be valid for **24 hours** after creation. This token will have the ability to be used for **any** of the API calls the user is authorized for within XIQ.

Graphical user interface, text, application

Description automatically generated

For the purpose of this script we will use this token to generate a sperate token with limited access and a specified expiration time. Copy the access token created not including the “”s.

##### Authorize in Swagger

At the top of the Swagger page, click the authorize button. A window will pop up allowing you to paste the access token. Clicking “Authorize” Swagger will set swagger to use the added access token for the API calls on the page.

Graphical user interface, application, Teams

Description automatically generated

##### Generating Specific Tokens



The */auth/apitoken* POST request allows you to specify an expire time as well as set permissions for a token. This is a great way to create a token for a specific application or script, only allowing the token to perform the needed tasks.

The expire time uses Epoch time, which is the number of seconds since midnight on Jan 1, 1970 (UTC). <https://www.epochconverter.com/> is a webpage that can be used to convert a readable time to epoch time or epoch time to a more readable time. Set a time for 1 year out and get the epoch time.

For this script we will want to have the following permissions - enduser, pcg:key   
This will give us access to view, create, and delete ppsk users as well as view, create, and delete pcg-key-based users if necessary.

Adding the desired expire time and a list of permissions, this API will return a token that will only be usable by the specified APIs.

*Request Body*

{  
 "description": "Token for XIQ-AD-PPSK-Sync.py script",  
 "expire\_time": 1628186428,  
 "permissions": [  
 "enduser”,  
 “pcg:key”  
 ]  
}

*Response Body*

**{**

**"access\_token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0aW1qc21pdGgyNEBwcm90b25tYWlsLmNvbSIsInNjb3BlcyI6WyJhY2NvdW50OnIiXSwidXNlcklkIjoyMTc5MjMyMSwicm9sZSI6IkFkbWluaXN0cmF0b3IiLCJjdXN0b21lcklkIjoyMTc5MTk3MSwiY3VzdG9tZXJNb2RlIjowLCJoaXFFbmFibGVkIjpmYWxzZSwib3duZXJJZCI6MTc5MTYxLCJvcmdJZCI6MCwiZGF0YUNlbnRlciI6IklBX0dDUCIsImlzcyI6ImV4dHJlbWVjbG91ZGlxLmNvbSIsImlhdCI6MTYyODE4MzA4OSwiZXhwIjoxNjI4MTg2NDI4fQ.CtBGq4YVGB9FzCodr6Oi5IG8yy1-4B-77AWl5rVG3S0",**

**"create\_time": "2021-11-29T15:47:57.000+0000",**

**"expire\_time": "2021-11-29T16:10:08.000+0000",**

**"creator\_id": 21792321,**

**"customer\_id": 21791971,**

**"description": "Token for XIQ-AD-PPSK-Sync.py script",**

**"permissions": [**

**"enduser",**

**"pcg:key"**

**]**

**}**

Copy the newly created access\_token and add it to the XIQ\_token variable in the script.

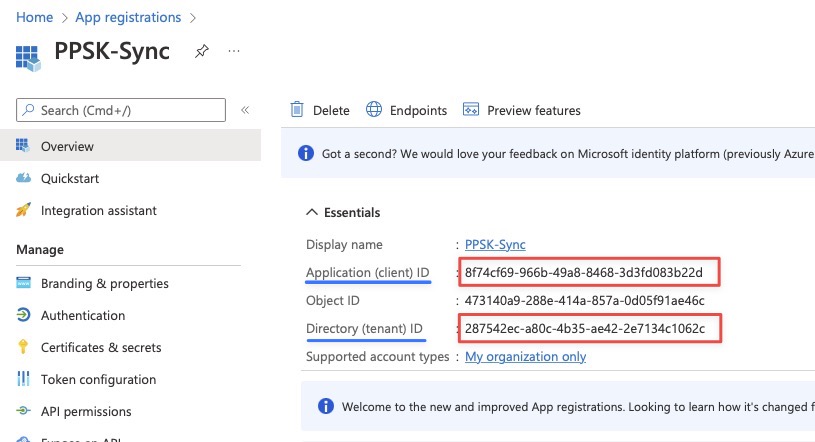
## Graphical user interface, application, Word Description automatically generatedA picture containing application Description automatically generatedCreating the Azure AD App

Graphical user interface, text, application, email

Description automatically generatedTo setup the Azure AD App, in the Azure portal select App registration under Azure services. This page will list any apps that you have. On the top select the option to add a **‘+ New registration’**.

This will open a page where you can give your application a name, select the account type, and register the app.

Once the app is registered, a screen will open with information about the app. Here you can get the **tenant\_id** and **client\_id**. Copy these items into the script.



##### Granting Permissions to the app

You will need to add permissions for the app to collect group information and user information. To do this, on the left, select **‘API Permissions’** then select **‘+ Add a permission’**.This will open a window on the right to request API permissions. In this window select the box for **‘Microsoft Graph’**.

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, text, application

Description automatically generatedIn the next window, select **‘Application permissions**.This will allow the application to run without redirecting for user authentication.

Graphical user interface, text, application

Description automatically generatedYou can then search for ‘group’, select **‘Group.Read.All’**, and click **‘Add permission’**.This will grant access for the Azure API to read group information.

You will also need to add permission for **‘User.Read.All’** to grant access for reading user information through the API.

Graphical user interface, text, application, email, Teams

Description automatically generatedFollow the same process of selecting **‘+ Add a permission’**, selecting the box for **‘Microsoft Graph’**, selecting **‘Application permissions’**, searching for ‘user’, select **‘User.Read.All’**, and clicking **‘Add permission’**.

Once the permissions are added, you will need to Grant admin consent for Default Directory. You can do this by selecting the checkmark next to the **‘Add a permission’** button.

Graphical user interface, text, application, Teams

Description automatically generated

##### Creating Client Secret

After the permissions are created and granted consent you can create the client secret. **NOTE:** If the client secret is created before the above process is completed a new secret will need to be created.

Graphical user interface, application, Teams

Description automatically generated

To create the Client Secret, select the **‘Certificates & secrets’** on the left. Select the **‘Client secrets’** tab, and select the **‘+ New client secret’** button.

A screen will open that will allow you to add a Description and set a time for ‘client secret’ to expire.

Graphical user interface, text, application

Description automatically generated

Once the client secret is created, you can copy the

Value that is created and add it to the script.

Graphical user interface, text, application

Description automatically generated

## AD Group Object ID

Graphical user interface, application

Description automatically generatedEach group in Azure AD will have a unique Object ID assigned to it. An easy way to get this ID is to log into your Azure portal, go into Azure Active Directory, and select Groups. In the groups section you can search for the group by name. In the list of groups, each group will have a unique Object ID displayed in the Object ID column.

Once the Object ID is obtained, add it into the group\_roles object in the script

group\_roles = [

# AD GROUP Object ID, XIQ group ID

(**"****e7274e17-d57f-4cb3-b3e5-fe7231f952e5",** "XIQ User Group ID"),

("AD Group Object ID", "XIQ User Group ID")

]

## XIQ User Group ID

Each XIQ User Group will be assigned a unique ID when it gets created. This is something that gets used by the backend systems and not seen in the GUI. The easiest way to get the ID is from the swagger page.

Go back to the swagger page and scroll down to the **Configuration – User Management** section and find the */usergroups* GET request.   
Click the “Try it out” button then the “execute” button. When you find the Name of the XIQ User Group you would like to use, it will be located inside of a pair of {curly brackets}. Inside the same pair of curly brackets will be an element called **id**. This is the ID that is needed.

*Response Body*

**{**

**"page": 1,**

**"count": 10,**

**"data": [**

**{**

**"id": 769490635824436,**

**"name": "Home\_Hive",**

**"description": "",**

**"predefined": false,**

**"create\_time": "2021-10-11T18:24:33.000+0000",**

**"update\_time": "2021-10-11T18:24:33.000+0000",**

**…**

Once that is obtained add it into the group\_roles object in the script.

group\_roles = [  
 # AD GROUP Object ID, XIQ group ID  
 ("e7274e17-d57f-4cb3-b3e5-fe7231f952e5", ", "**769490635824436**"),  
 ("AD Group Object ID", "XIQ User Group ID")  
]

Continue to gather other AD Group Distinguished Names and XIQ User Group IDs if needed. Enter them in the same format with each set enclosed in parentheses. All but the last one should be followed by a coma.

group\_roles = [  
 # AD GROUP Object ID, XIQ group ID  
 ("d1ba206a-0213-4ae1-95de-0778b366b778", "769490635824438"),  
 ("e7274e17-d57f-4cb3-b3e5-fe7231f952e5", "769490635824436"),  
 ("27a8d84c-f4c4-41e2-a84e-9abdac7c0c13", "769490635824395")  
]

## Graphical user interface, text, application, email Description automatically generatedXIQ Network Policy ID

Each XIQ Network Policy will be assigned a unique ID when it gets created. The easiest way to get the ID is to select the Network Policy in the XIQ GUI. When you select the Network Policy, the ID is the long number listed in the URL. The Policy Name is directly under the Policy Details.

You can also get the Network Policy ID from Swagger for all Network Policies with PCG configured.

Go back to the swagger page and scroll down to the **Configuration – User Management** section and find the */pcg/key-based* GET request. If the Network Policy is configured to use PCG, it will be listed in the response. The policy id and policy name are included in this response.



*Response Body*

**[**

**{**

**"id": 1059916324374423,**

**"create\_time": "2021-12-13T15:37:55.000+0000",**

**"update\_time": "2022-01-06T20:29:15.000+0000",**

**"org\_id": 0,**

**"policy\_id": 1059916324374274,**

**"policy\_name": "JB\_Lab",**

**"ssid\_name": "PCG\_Test",**

**"enabled": true,**

**"users": [**

**{**

# Running the Script:

To run the script, open the terminal (PowerShell for Windows) to the location of the script and run:

Python3 XIQ-Azure-AD-PPSK-Sync.py

chmod +x XIQ-Azure-AD-PPSK-Sync.py

You can also make the script executable by running

Then you can run the script by typing

./XIQ-Azure-AD-PPSK-Sync.py

The script will print to the screen how many PPSK users were parsed and how many AD users were parsed. If there are any users in the list of AD users and not in the list of PPSK users, an API call will be made to create the PPSK user. The script will print on the screen for each user it successfully creates.

successfully created PPSK user Tim Smith

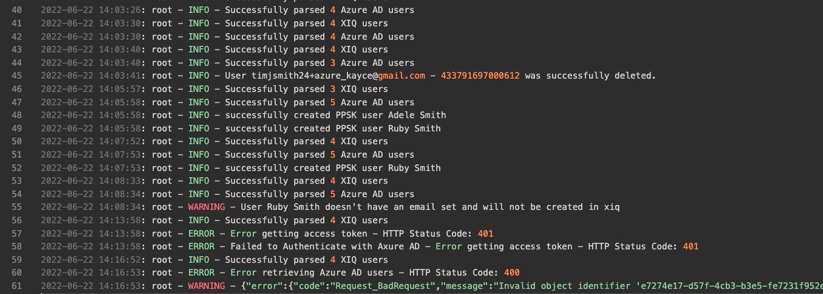
If there are any users in the list of PPSK users that are not in the list of AD users or in the list of disabled AD users, an API call will be made to delete the PPSK user. The script will print on the screen for each user it successfully deletes. – This will be the email address of the user.

User user0200@example.com - 769490635839948 was successfully deleted.

The AD user needs to have an email address assigned or the PPSK user will not be created. A message will print on the screen.

User Ruby Smith doesn't have an email set and will not be created in xiq

## Log File

Upon running the script, a log file will be created named XIQ-Azure-AD-PPSK-Sync.log. Additional runs of the script will append to this log file. This file will contain this same information that prints to the screen as well as any error received when making the API calls. This is a good place to look if issues are seen.

# Scheduling Script to Run

## Mac & Linux based Systems

A Cron job can be setup to automatically run the script at a specified interval. Ideally this could be set for every 8, 12, or 24 hours. This would make sure the AD group and PPSK user group would stay in sync. The script can also be ran manually between those times if a user needed to be added right away or removed right away.

### Setting up a Cron Job

Open and edit the crontab and configure the job with arrangement for the command you want to run. From the terminal window enter the following command.

crontab -e

There are 3 parts to a cron job configuration.

#### Cron Job Time Format

Part 1 of the cron job

The first 5 characters a b c d e represent the time, date, and repetition of the job

a – Minute (0-59)   
b – Hour (0-23)

c – Day (0-31)

d – Month (0-12) – 0=None and 12 = December

e – Day of the Week (0-7) – 0=Sunday and 7=Sunday

* **An asterisk (\*)** stands for all values. Use this operator to keep tasks running during all months, or all days of the week.
* **A forward-slash (/)** is used to divide a value into steps. (\*/2 would be every other value, \*/3 would be every third, \*/10 would be every tenth, etc.)

To set the Cron job to run every 8 hours the time format would look like this.

0 \*/8 \* \* \*

To set the Cron job to run every night at midnight the time format would look like this.

0 0 \* \* \*

#### Cron Job Script and Script Location

Part 2 of the cron job

The next part is where you enter the script that you want to run and its location

python3 /home/admin/documents/scripts/XIQ-Azure-AD-PPSK-Sync.py

You can make the script executable, so you don’t have to type python3 before entering the script. Instead, you will just enter a period before the location.

chmod +x /home/admin/documents/scripts/XIQ-Azure-AD-PPSK-Sync.py

./home/admin/documents/scripts/XIQ-Azure-AD-PPSK-Sync.py

#### Cron Job Output and Job Completion

Optional Part 3 of the cron job

The last part is an optional part that specifies where the output and completion of the script should go. If not specified, the cron will send an email to the owner of the crontab file.

To avoid filling up the inbox on the server, it is recommended to have something set for the output. This can be set to append to a file

>> /home/admin/documents/scripts/XIQ-Azure-AD-PPSK-Sync-Output.txt

Or can be set just turn off the email output

> /dev/null 2>&1

#### 

#### Cron Job Command Example

The command should be entered in a single line and then save the crontab file.

*Every 8 hours turning off output*

0 \*/8 \* \* \* python3 /home/admin/documents/scripts/XIQ-Azure-AD-PPSK-Sync.py > /dev/null 2>&1

*Every 12 hours with saved output*

0 \*/12 \* \* \* ./home/admin/documents/scripts/XIQ-Azure-AD-PPSK-Sync.py >> /home/admin/documents/scripts/XIQ-Azure-AD-PPSK-Sync-Output.txt

## Windows based Systems

A Windows task schedule can be setup to automatically run the script at a specified interval. Ideally this could be set for every 8, 12, or 24 hours, this would make sure the AD group and PPSK user group would stay in sync. The script can also be executed manually between those times if a user needed to be added right away or removed right away.

### Setting up Windows Task Scheduler

Open Control Panel > System and Security > Administrative Tools > Task Scheduler

Selected ‘Create basic task…’

Give you task a name like ‘AD-PPSK-Sync’ and click ‘Next’

Leave the Trigger set to daily and click ‘Next’ – We will come back and adjust this

Click ‘Next’ leaving recur every 1 day

Select ‘Start a program and click ‘Next’

#### Start a Program

For the Program/script: section enter the path of your python.exe file.

Graphical user interface, text, application, email

Description automatically generatedThe location of your python.exe file depends on how it was installed. An easy way to find this location is to open Windows PowerShell and enter python3. This will open the python interpreter. In the interpreter enter then

sys.executable

Import sys

exit()

This will output the location of the python.exe file. Enter to exit the interpreter.

Enter the full path of the python.exe file in the Program/Script: field.

Graphical user interface, text, application

Description automatically generated

In the Add arguments (optional): field, enter the name of the script.

XIQ-Azure-AD-PPSK-Sync.py

In the Start in (optional): field, enter the location of the script.

C:\user\your\_python\_project\_path

#### Editing the Time

Once the task is saved, open the Task Scheduler Library folder and find the newly created AD-PPSK-Sync task. Click on it to open and select the Trigger tab and edit the Daily trigger. Here you can set what time you want it to run.

If you want the script to run every 8 or 12 hours check the box next to ‘Repeat task every:’ and enter ‘*8 hours’* or *’12 hours’*

Graphical user interface, application

Description automatically generated

# Troubleshooting:

## Log File

The XIQ-Azure-AD-PPSK-sync.log file is a good place to look for potential issues. This log file will update whenever the script is ran, whether manually or on a schedule.

### Invalid XIQ token

2021-11-05 16:58:27: root - ERROR - Error retrieving PPSK users from XIQ - HTTP Status Code: 401

2021-11-05 16:58:27: root - WARNING - {'error\_code': 'AuthInvalidToken', 'error\_id': 'cda656a5157d4c87a5143252aad71bff', 'error\_message': 'Unable to read JSON value: ?[\x19???\x14?M??'}

Check token using [Swagger](#_Swagger_1) - remember that if you generated a specific token it may only have access to the user APIs.

### Invalid XIQ token format

2021-11-08 13:56:36: root - ERROR - Error retrieving PPSK users from XIQ - HTTP Status Code: 401

2021-11-08 13:56:36: root - WARNING - {'error\_code': 'AuthInvalidToken', 'error\_id': '555d1ce9f67b40ef83caf4a89ca92b04', 'error\_message': 'JWT strings must contain exactly 2 period characters. Found: 0'}

This may mean that you are trying to use XIQ Username and Password but did not comment out line 31. Add a # in front of **line 31**. Or the token wasn’t entered correctly

### Expired XIQ token – Code 401 & JWT expired

2021-11-08 14:14:16: root - ERROR - Error retrieving PPSK users from XIQ - HTTP Status Code: 401

2021-11-08 14:14:16: root - WARNING - {'error\_code': 'AuthTokenExpired', 'error\_id': '78d8b818a03940dd8d4accfc1b3ffb7e', 'error\_message': 'JWT expired at 2021-11-08T19:14:11Z.

The XIQ token has expired. Generate a new token using [Swagger](#_Swagger_1)

### Invalid XIQ Username/password – Code 401

2021-11-08 13:55:15: root - ERROR - Error getting access token - HTTP Status Code: 401

2021-11-08 13:55:15: root - WARNING - <Response [401]>

Check username and password for XIQ. – recommend using XIQ Token

### Failed Authentication for Azure AD API

2022-06-22 14:13:58: root - ERROR - Failed to Authenticate with Axure AD - Error getting access token - HTTP Status Code: 401

Check the tenant id, client id, and client secret**.**

### Invalid or Wrong Group ID entered

2022-06-22 14:16:53: root - ERROR - Error retrieving Azure AD users - HTTP Status Code: 400

2022-06-22 14:16:53: root - WARNING - {"error":{"code":"Request\_BadRequest","message":"Invalid object identifier 'e7274e17-d57f-4cb3-b3e5-fe7231f952e'.","innerError":{"date":"2022-06-22T18:16:53","request-id":"353bdc25-3d80-4e6a-9f6f-c2c205707f0a","client-request-id":"353bdc25-3d80-4e6a-9f6f-c2c205707f0a"}}}

Check the group ID for the Azure AD group

### XIQ User Failed to Create – Code 400

2021-11-08 14:19:35: root - INFO - Successfully parsed 0 XIQ users

2021-11-08 14:19:36: root - INFO - Successfully parsed 4 LDAP users

2021-11-08 14:19:36: root - ERROR - Error adding PPSK user Tim Smith - HTTP Status Code: 400

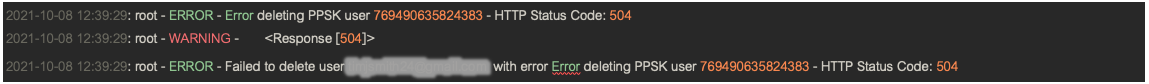
2021-11-08 14:19:36: root - WARNING - {'error\_code': 'UNKNOWN', 'error\_id': None, 'error\_message': 'UNKNOWN'}

2021-11-08 14:19:36: root - ERROR - failed to create Bauer Smith: Error adding PPSK user Tim Smith - HTTP Status Code: 400

There are a couple possibilities when you see this error. As you can see above there were 0 XIQ users parsed, and the user failed to create. In this instance the XIQ User Group ID was incorrect. If XIQ users parsed was 0 and you have configured users in the user group check the user group ID in the group\_roles list on **Lines 33-37**

The other thing that could cause this Error/Warning is if the username already exists in XIQ PPSK users within a different user group.

### XIQ Timeout Error



2021-10-08 12:43:05: root - ERROR - Failed retrieving PPSK users from XIQ - HTTP Status Code: 504

2021-10-08 12:43:05: root - WARNING - <Response [504]>

A HTTP Status Code: **504** is a timeout from XIQ. If XIQ cannot respond to the API call within 60 secs it will send these **504** errors. The script will need to be ran again with no changes.

### Email is not set for AD User

2021-11-08 14:42:39: root - WARNING - User Tim Smith doesn't have an email set and will not be created in xiq

Check the AD User and see if email is set. Email is required to create a PPSK.

### Intermittent API issue

This is a known issue when making hundreds of API calls consecutively. This is something that will be fixed soon. This may be seen when hundreds of PPSK users are being created or deleted in a run of the script. If this is seen, run the script a second time and it will get the couple of stragglers left behind.

2022-02-17 15:51:29: root - ERROR - Error adding PPSK user User0975 Last - HTTP Status Code: 400

2022-02-17 15:51:29: root - WARNING - {'error\_code': 'UNKNOWN', 'error\_id': 'e1ff18a4ad7f47148bae36d6ca943715', 'error\_message': 'UNKNOWN: Either vhmId or vhmAccountId must be present in HiveContext; nested exception is java.lang.IllegalArgumentException: Either vhmId or vhmAccountId must be present in HiveContext'}

2022-02-17 15:51:29: root - ERROR - failed to create User0975 Last: Error adding PPSK user User0975 Last - HTTP Status Code: 400

## Azure\_AD\_Test.py

This script was written to help troubleshoot issues with collecting data from AD. The variables are the same in this script. Except the AD Group ID should be added to the **ad\_group\_id** variable instead of in the group\_roles list.

This script will test the Azure AD app API. If there are any errors, they will print to the screen. Otherwise, the collected data from the AD server will print on screen. If empty [] brackets print, check the [Group ID](#_AD_Group_Object) and validate that it is correct.

Using this output, you can check if the user is disabled users or not in Azure AD. You can validate there is an email set for the user. And overall check that information is being returned.

completed page of AD Users. Total Users collected is 5

Tripp Smith {'accountEnabled': True, 'email': 'user0001@example.com', 'username': 'Tripp@example.onmicrosoft.com'}

Miles Smith {'accountEnabled': True, 'email': 'user0002@example.com', 'username': 'Miles@example.onmicrosoft.com'}

Bauer Smith {'accountEnabled': False 'email': 'user0003@example.com', 'username': 'Bauer@example.onmicrosoft.com'}

Adele Smith {'accountEnabled': True, 'email': ‘user0004@example.com', 'username': 'Adele@example.onmicrosoft.com'}

Ruby Smith {'accountEnabled': True, 'email': None, 'username': 'rubysmith@example.onmicrosoft.com'}