

Vertiv™ Avocent® RM1048P Rack Manager and Vertiv™ Avocent® MP1000 Management Platform

Using the Viewer Session API Technical Note

VIEWER SESSION API v1.0 and v1.1, NOVEMBER 2023

Technical Note Section Outline

1. Overview
2. Launching Viewer Sessions
3. Python Script Example

1. Overview

NOTE: At this time, the former Vertiv™ Avocent® ADX platform is transitioning into the Vertiv™ Avocent® DSView™ solution. During this transition, there may temporarily still be references to “ADX” within product-related features and documentation.

This technical note provides instructions on launching viewer sessions (KVM, serial and virtual, for example) to target devices in the Vertiv™ Avocent® DSView™ solution by using the Application Programming Interfaces (APIs) within the solution. The instructions within this technical note apply to the Vertiv™ Avocent® MP1000 Management Platform appliance, the Vertiv™ Avocent® MP1000VA Management Platform Virtual Appliance, and the Vertiv™ Avocent® RM1048P Rack Manager appliance.

IMPORTANT NOTE: This technical note assumes that you are already familiar with API operations within the Vertiv™ Avocent® DSView™ solution and have a basic knowledge of Python programming language. Additional information on the APIs within the Vertiv™ Avocent® DSView™ solution is available in the Vertiv™ Avocent® DSView™ Solution API Guide (see the Session Management section) available here: [API Guide](#)

Supported Platform Version Information

- In July 2023, API v1.0 was released to provide the initial support for launching KVM, serial and virtual sessions. This version supports the following product versions:
 - Vertiv™ Avocent® MP1000 Management Platform version 3.33.5 or higher
 - Vertiv™ Avocent® MP1000VA Management Platform Virtual Appliance version 3.33.5 or higher
 - Vertiv™ Avocent® RM1048P Rack Manager version 1.33.6 or higher
- In August 2023, API v1.1 was released to add support for launching Standalone Passive KVM sessions. This version supports the following product versions:
 - Vertiv™ Avocent® MP1000 Management Platform version 3.33.8 or higher
 - Vertiv™ Avocent® MP1000VA Management Platform Virtual Appliance version 3.33.8 or higher

2. Launching Viewer Sessions

The Viewer Session API is a URL-based API that enables you to launch KVM, serial and Virtual Machine sessions. Once you know the device ID and the type of session you need, you can quickly integrate with third party applications (such as a help desk or ticketing system) that require technicians to launch remote sessions for troubleshooting and system recovery.

Prior to launching a viewer session, you must complete the following steps:

- Create a user login session
- Identify information for the target devices available in the Vertiv™ Avocent® DSView™ solution, including the types of viewer sessions supported by each device
- Specify the target device programmatic name
- Create a viewer session ticket

- Create a URL to launch the viewer session

Procedures for completing each of these steps are available in the following sections.

NOTE: A Python script example is available in section 3 of these release notes. This script example may be modified as needed to perform the procedures in the following sections and/or to launch different types of viewer sessions.

Creating a User Login Session

To create a user login session:

1. Log into your Vertiv™ Avocent® DSView™ solution appliance REST API with one of the following addresses:
 - For the Vertiv™ Avocent® MP1000 Management Platform appliance, enter **https://<MP_1000_IP_ADDRESS>/api/v1/userSessions**.
 - For the Vertiv™ Avocent® MP1000VA Virtual Appliance, enter **https://<MP_1000_VA_IP_ADDRESS>/api/v1/userSessions**.
 - For the Vertiv™ Avocent® RM1048P Rack Manager appliance, enter **https://<RM_1048P_IP_ADDRESS>/api/v1/userSessions**.
2. Use the details in the following table to create a user login session in the API. A sample API response is also provided for reference after the table.

NOTE: The Response Values listed in the table are only a subset of the values available and relevant to launching viewer sessions.

REST API METHOD	REST API CALL	BODY PARAMETERS	RESPONSE VALUES
POST	/api/v1/userSessions	<p>username: The name of a user in the Vertiv™ Avocent® DSView™ solution with sufficient privileges to launch Viewer sessions.</p> <p>password: The user password.</p>	<p>The HTTP status code must be 201.</p> <p>user_session: The user login session information (includes the user session name, user session id and login username). The user session id will be used in a subsequent REST API call to create a unique ticket id for launching a viewer session.</p> <p>jwt: The JWT token associated with the user login session. The JWT token is used in the Authorization header field for subsequent REST API calls.</p>

Response Sample

```
{
  "user_session": {
    "name": "userSessions/1d4d522b-812f-4c05-9c57-b88a4beb133d",
    "id": "1d4d522b-812f-4c05-9c57-b88a4beb133d",
    "username": "admin"
  },
  "jwt": "eyJhbGciOiJIUzU1NiIsImtpZCI6IjE6ImE4NGVmZmYyZmIOMGE..."
}
```

Identifying Target Device Information and Specifying the Target Device Programmatic Name

To retrieve information on a target device within the Vertiv™ Avocent® DSView™ solution:

- Use one of the following addresses to retrieve information on a target device:
 - For the Vertiv™ Avocent® MP1000 Management Platform appliance, enter **https://<MP_1000_IP_ADDRESS>/api/v1/devices**.
 - For the Vertiv™ Avocent® MP1000VA Virtual Appliance, enter **https://<MP_1000_VA_IP_ADDRESS>/api/v1/devices**.
 - For the Vertiv™ Avocent® RM1048P Rack Manager appliance, enter **https://<RM_1048P_IP_ADDRESS>/api/v1/devices**.
- In order to search for the viewer session type supported by a target device, you must enter a URL that specifies the target device programmatic name. The following table provides a list of the target device programmatic names for each type of viewer session and target device type.

VIEWER SESSION TYPE	TARGET DEVICE TYPE	TARGET DEVICE PROGRAMMATIC NAME	CONNECTION PATH TYPE
KVM	IPIQ	ADX-IP-IQ	
	IPUHD	ADX-IP-UHD	DCP_KVM_IP
	ACS Serial Target	SERIAL-TARGET	
SERIAL	ACS Appliance	ACS	
	IPUHD	ADX-IP-UHD	DCP_SERIAL_IP
	IPSL	ADX-IP-SERIAL-PORT	
VM	Virtual Machine	VS-VIRTUAL-MACHINE	

Before plugging in the target device programmatic name into the applicable URL address, ensure you are aware of the following requirements:

- The Viewer Session Type, Target Device Programmatic Name and Connection Path Type values must be specified in all uppercase letters as shown in the table.
- The Vertiv™ Avocent® IPUHD 4K IP KVM device supports both KVM and serial viewer sessions; the target device programmatic name is the same value for both (ADX-IP-UHD). For KVM viewer sessions, the DCP_KVM_IP Connection Path Type value must be verified before launching a KVM viewer session. The Connection Path Type value must also be verified before launching a serial viewer session (DCP_SERIAL_IP).
- Prior to launching a Virtual Machine session, enter **https://<VM_SERVER_IP>** into a browser. The <VM_SERVER_IP> is the IP address of the VM Server hosting the Virtual Machine. When prompted to accept the VM Server, you **MUST** accept the certificate. You may optionally close the browser window.

Enter the applicable URL with the appropriate target device programmatic name:

- For the Vertiv™ Avocent® MP1000 Management Platform appliance, enter:
https://<MP_1000_PLATFORM_IP_ADDRESS>/api/v1/devices?filter=device_type_programmatic_name<TARGET_DEVICE_PROGRAMMATIC_NAME>
- For the Vertiv™ Avocent® MP1000VA Virtual Appliance, enter:
https://<MP_1000VA_PLATFORM_IP_ADDRESS>/api/v1/devices?filter=device_type_programmatic_name<TARGET_DEVICE_PROGRAMMATIC_NAME>
- For the Vertiv™ Avocent® RM1048P Rack Manager appliance, enter:
https://<RM_1048P_IP_ADDRESS>/api/v1/devices?filter=device_type_programmatic_name<TARGET_DEVICE_PROGRAMMATIC_NAME>

- Use the details in the following table to get information on a specific target device within the Vertiv™ Avocent® DSView™ solution. A sample API response is also provided for reference after the table.

NOTE: The Response Values listed in the table are only a subset of the values available and relevant to launching viewer sessions. Additional details are available in the Vertiv™ Avocent® DSView™ Solution API Guide located here: [API Guide](#)

REST API METHOD	REST API CALL	HEADER FIELDS	RESPONSE VALUES
GET	/api/v1/devices?filter=device_programmatic_name<TARGET_DEVICE_PROGRAMMATIC_NAME>	<p>Authorization: This field must include the JWT token corresponding to an existing user login session in the Vertiv™ Avocent® DSView™ solution. The type of authorization must be a Bearer Token.</p>	<p>The HTTP status code must be 200.</p> <p>devices: This array list contains information on the target devices within the platform. You can search for the device name of the target device in this array list for which you want to launch a viewer session. Target device names are case sensitive. After the target device is found, you may retrieve the device id from the id field.</p> <p>device id: The device id is a unique identifier that is associated with a target device within the Vertiv™ Avocent® DSView™ solution. This device id is used as a parameter in the URL to launch a viewer session to a target device.</p>

Response Sample: Vertiv™ Avocent® IPIQ IP KVM Device Target Device for KVM Viewer Session

```
{
  "devices": [
    {
      "id": "11eb0467-5022-45b6-8b69-
        dee9f33ad39d",
      "device_name": "IPIQ Target Device ",
      "device_type_programmatic_name":
        "ADX-IP-IQ",
    }, ...
  ]
}
```

Response Sample: Vertiv™ Avocent® IPUHD 4K IP KVM Device Target Device for KVM Viewer Session

```
{
  "devices": [
    {
      "id": "c30cf052-dfd2-463c-baf7-
        d03885361f71",
      "device_name": "IPUHD Target Device ",
      "device_type_programmatic_name":
        "ADX-IP-UHD",
      "connection_path_info": [
        {
          "path_type": "DCP_KVM_IP", ...
        }, ...
      ], ...
    }, ...
  ]
}
```

Creating a Viewer Session Ticket

NOTE: For the following procedure, you will need the user session id (USER_SESSION_ID) that you received when creating a user login session earlier in this technical note.

To create a viewer session ticket:

- Use one of the following addresses to create a viewer session ticket:
 - For the Vertiv™ Avocent® MP1000 Management Platform appliance, enter:
https://<MP_1000_IP_ADDRESS>/api/v1/userSessions/<USER_SESSION_ID>:createTicket
 - For the Vertiv™ Avocent® MP1000VA Virtual Appliance, enter:
https://<MP_1000_VA_IP_ADDRESS>/api/v1/userSessions/<USER_SESSION_ID>:createTicket
 - For the Vertiv™ Avocent® RM1048P Rack Manager appliance, enter:
https://<RM_1048P_IP_ADDRESS>/api/v1/userSessions/<USER_SESSION_ID>:createTicket
- Use the details in the following table to create a viewer session ticket in the API. A sample API response is also provided for reference after the table.

NOTE: The Response Values listed in the table are only a subset of the values available and relevant to launching viewer sessions. Additional details are available in the Vertiv™ Avocent® DSView™ Solution API Guide located here: [API Guide](#)

REST API METHOD	REST API CALL	HEADER FIELDS	RESPONSE VALUES
POST	/api/v1/userSessions/<USER_SESSION_ID>:createTicket	<p>Authorization: This field must include the JWT token corresponding to an existing user login session in the Vertiv™ Avocent® DSView™ solution. The type of authorization must be a Bearer Token.</p>	<p>The HTTP status code must be 200.</p> <p>ticket: A unique ticket id that can be used in the URL for launching a single viewer session to a target device in the platform. This ticket id is used as a parameter in the URL to launch a viewer session to a target device.</p> <p>IMPORTANT NOTE: The ticket id is only valid for 20 seconds and MUST NOT be reused for launching any other viewer sessions.</p>

Response Sample

```
{
  "ticket": "2db72e02f4d414acf3e7667e
            8b2b32c36c9d2511"
}
```

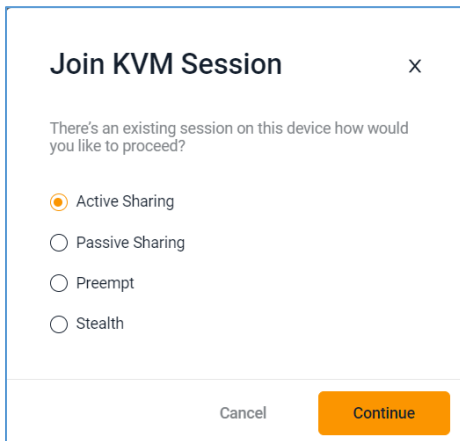
Creating a URL to Launch a Viewer Session

In order to launch a viewer session to a target device, you must first create a final URL. Once defined, the URL is entered into a browser to launch a session. The following information must be defined so that you can insert it into the URL:

- The <VIEWER_TYPE>, which must be all uppercase letters:
 - KVM (to launch a KVM viewer session)
 - SERIAL (to launch a serial viewer session)
 - VM (to launch a Virtual Machine viewer session)
- The <DEVICE_ID>, which is the target device id that you obtained in the previous Identifying Target Device Information... section (see step 3 in that section).
- The <TICKET_ID>, which is the ticket id created in the previous Creating a Viewer Session Ticket section (see step 2 in that section).

If you are launching a KVM viewer session, you have some additional options that can be added into the URL:

- Sharing mode: Adding **&sharingMode=<SHARING_MODE>** to the URL prevents the dialog box with sharing options from being displayed. If the sharing options dialog box displays (see the following example), the sharing mode can be changed.

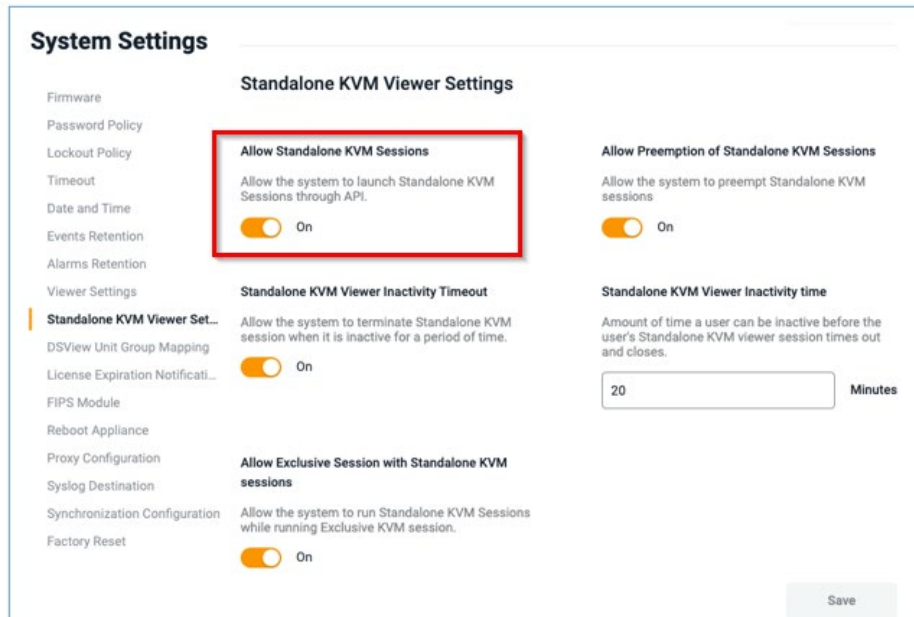


The <SHARING_MODE> options that can be added are listed in the following table.

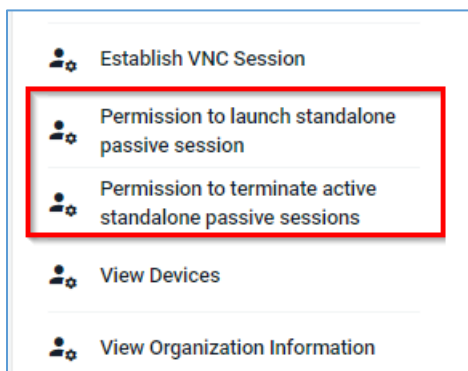
SHARING MODE	DESCRIPTION
ACTIVE	The secondary viewer session can share the primary viewer session to the target device.
PASSIVE	The secondary viewer session can share the primary viewer session in passive (read only) mode.
PREEMPT	The secondary viewer session can preempt all existing viewer sessions to the target device.
STEALTH	The secondary viewer session can share the primary viewer session in stealth mode.
STANDALONE_PASSIVE	This is a read-only KVM session in which the user is unable to interact with the mouse and keyboard. In this mode, some features (including virtual media) are not available. A Standalone Passive KVM session will not receive a sharing request when users are trying to establish a KVM session to the same target device. See the following procedure to set up Standalone Passive KVM Session mode.

To set up Standalone Passive KVM Mode:

1. Log into the Vertiv™ Avocent® MP1000 Management Platform web User Interface (UI).
2. In the sidebar, select *Administration - System Settings*, then select *Standalone KVM Viewer Settings*.



3. Click the On setting in the Allow Standalone KVM Sessions section to enable the option, then click Save.
4. In the sidebar, select *Administration - Roles & Permissions*.
5. Select a role, then access the Permissions panel on the right-side of the screen. Ensure that permissions to launch and terminate standalone passive sessions are available to the role.



- Mouse mode: Adding **&mouseMode=<MOUSE_MODE>** to the URL sets the mouse mode after launching the viewer session. The following table lists the valid mouse modes available.

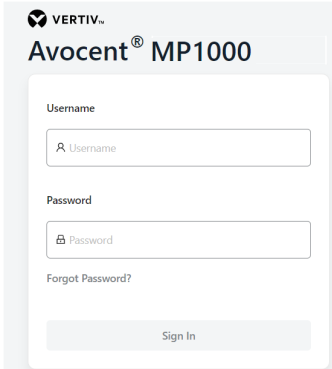
MOUSE MODE	DESCRIPTION
0	Use Absolute Mouse in the viewer session (default).
1	Use Relative Mouse in the viewer session.
2	Use Relative Mouse without mouse acceleration in the viewer session. NOTE: This only applies to Vertiv™ Avocent® IPUHD devices.

With all the URL information defined, you are ready to launch a viewer session.

Launching a Viewer Session to a Target Device

To launch a viewer session to a target device:

NOTE: If the ticket id is not specified in the URL, you are redirected to the platform login page (see the following example login screen). After the login authentication is successful, the viewer session is automatically launched.



Now that you have defined the values needed for your URL, use one of the following addresses to launch a viewer session to a target device:

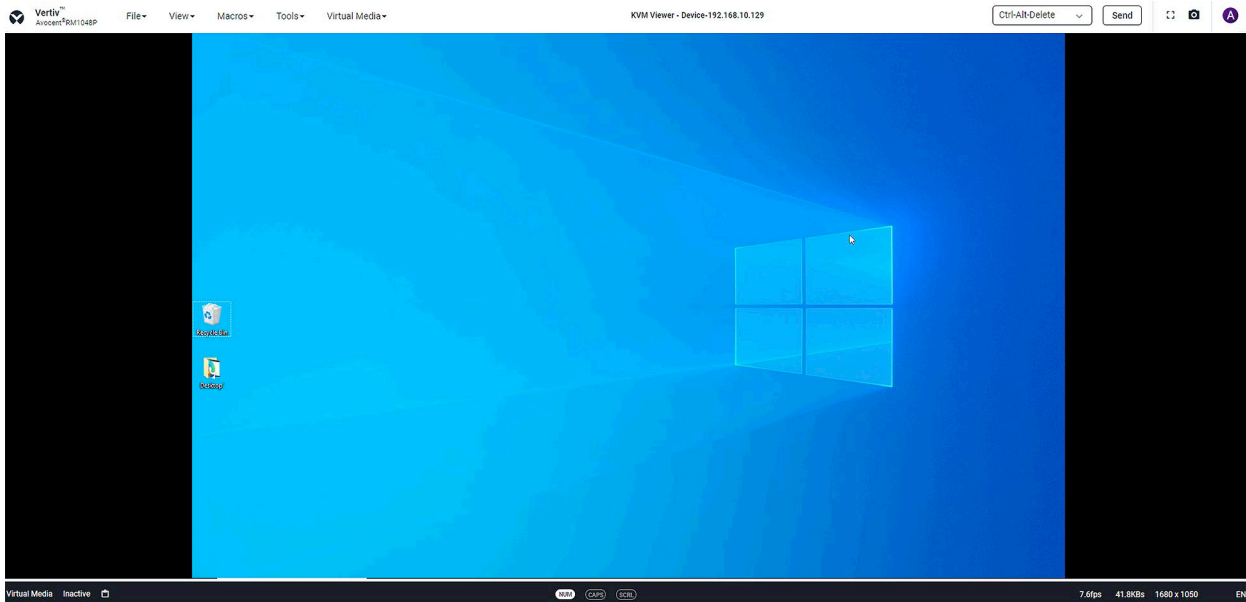
- For the Vertiv™ Avocent® MP1000 Management Platform appliance, enter:
`https://<MP_1000_IP_ADDRESS>/api/v1/launchUI/?launchType=viewer&viewerType=<VIEWER_TYPE>&deviceid=<DEVICE_ID>&ticket=<TICKET_ID>`
- For the Vertiv™ Avocent® MP1000VA Virtual Appliance, enter:
`https://<MP_1000_VA_IP_ADDRESS>/api/v1/launchUI/?launchType=viewer&viewerType=<VIEWER_TYPE>&deviceid=<DEVICE_ID>&ticket=<TICKET_ID>`
- For the Vertiv™ Avocent® RM1048P Rack Manager appliance, enter:
`https://<RM_1048P_IP_ADDRESS>/api/v1/launchUI/?launchType=viewer&viewerType=<VIEWER_TYPE>&deviceid=<DEVICE_ID>&ticket=<TICKET_ID>`

The viewer session launches to the target device.

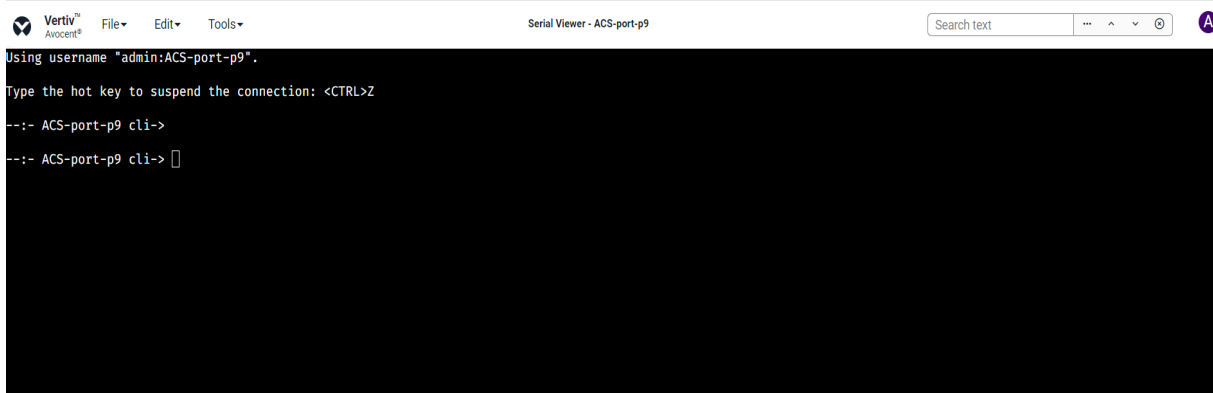
NOTE: If the browser displays certificate warnings, you can accept them.

The following examples show different types of viewer sessions and sample URLs for each:

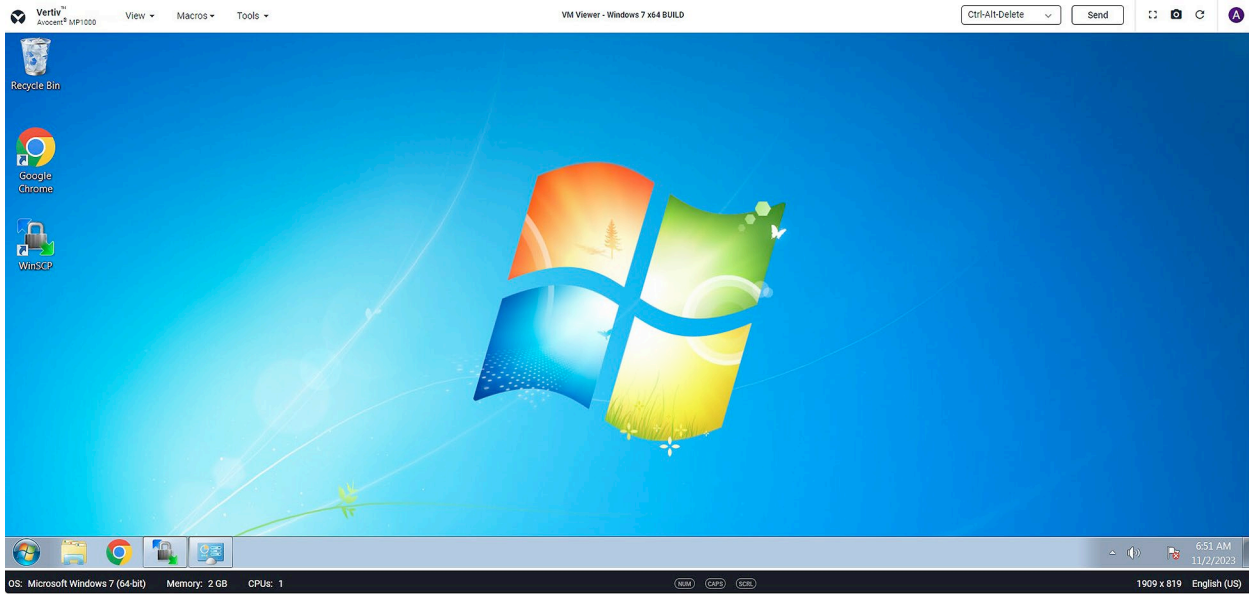
- Example KVM viewer session to a target device connected to a Vertiv™ Avocent® IPIQ IP KVM device:
 - URL sample: <https://10.200.15.43/api/v1/launchUI/?launchType=viewer&viewerType=KVM&deviceId=11eb0467-5022-45b6-8b69-dee9f33ad39d &ticket=2db72e02f4d414acf3e7667e8b2b32c36c9d2511>



- Example SERIAL viewer session to a target device connected to a Vertiv™ Avocent® ACS advanced console system appliance serial port:
 - URL sample: <https://10.200.15.43/api/v1/launchUI/?launchType=viewer&viewerType=SERIAL&deviceId=c30cf052-dfd2-463c-baf7-d03885361f71 &ticket=421b72f02f4d414acf3f7667e8b2b32e36c9d2783>



- Example VM viewer session to a target device:
 - URL sample: <https://10.200.15.43/api/v1/launchUI/?launchType=viewer&viewerType=VM&deviceId=d6acf052-dab2-433c-bcf7-d03895361b13 &ticket=561c72f02f4d414acf3f7667e8b2b32e36c9d2980>



3. Python Script Example

The following Python script provides sample code that you can modify as needed to launch viewer sessions. This particular example demonstrates how to launch a Standalone KVM Passive session to a target device from a Windows or Linux machine.

NOTE: Before running the Python script, ensure that you have previously installed Python version 3.8 or higher and any other required libraries specified in the script.

```
# Make sure that these libraries are already installed on the machine
import requests
import urllib3
import webbrowser
import getpass

# For demo purposes only, disable any certificate warnings on the browser. Note that the DSView
# solution appliance uses self-signed certificates by default. If certificate verification is not disabled,
# you will see several warning messages similar to the following:
#
# "InsecureRequestWarning: Unverified HTTPS request is being made to host"
#
# If you want to see these certificate warnings, you can remove or comment out the line below.
urllib3.disable_warnings()

# The number of seconds to wait for a REST API call to complete
TIMEOUT_SEC = 60.0

# STEP 1 - Login to the DSView solution appliance.

# Ask the user for the DSView solution appliance host IP address to connect
Host = str(input('Enter MP1000, MP1000VA or RM1048 IP address : '))
Host = 'https://' + Host
print('Login to the DSView solution appliance with IP Address: ', Host)

# Ask the user for the username and password to login to the DSView solution appliance
userName = str(input(('username: ')))
password = getpass.getpass('password: ')

# The REST endpoint to login to the DSView solution appliance
loginURL = '/api/v1/userSessions'

# Invoke REST API call to login to the DSView solution appliance
try:
    response = requests.post(Host + loginURL,
                             json={'username': userName, 'password': password},
                             timeout=TIMEOUT_SEC,
                             verify=False)
except (requests.exceptions.ConnectTimeout, requests.exceptions.SSLError):
    print(f'Unable to login to DSView solution appliance using IP address: "{Host}")')
    exit()
```

```
# Make sure the REST API call was successful
if response.status_code == 201:
    print('Login to the DSView solution appliance was successful.')

    # Get JWT token from the REST API call response
    json_response = response.json()
    if 'jwt' not in json_response:
        print('Authentication response did not contain jwt.')
    jwt = json_response['jwt']

    # Get the user login session ID from the REST API call response
    sessionId = json_response['user_session']['id']
    print('SessionId', sessionId)
else:
    print('Unable to login to the DSView solution appliance. HTTP Response code is ', response.status_code)
    exit()

# STEP 2 - Identify the target device in the DSView solution appliance to launch a Viewer session
print('Identify the target device in the DSView solution appliance to launch a Viewer session')

# The REST endpoint to get the list of IPIQ devices in the DSView solution appliance.
devicesURL = '/api/v1/devices'

# Specify the device type programmatic name that is associated with the type of target
# device for launching a viewer session. These are the valid values for the device programmatic name:
#
# ADX-IP-IQ : Launch a KVM viewer session to an IPIQ device.
# ADX-IP-UHD : Launch a KVM viewer session to an IPUHD device. You must specify the connection path type as well
# as described below:
#   DCP_KVM_IP: This value is needed to launch a KVM viewer session to an IPUHD target device.
#   DCP_SERIAL_IP: This value is needed to launch a Serial viewer session to an IPUHD target device.
# SERIAL-TARGET : Launch a SERIAL viewer session to a device connected to an ACS port.
# ACS : Launch a SERIAL viewer session to an ACS appliance.
# ADX-IP-SERIAL-PORT: Launch a SERIAL viewer session to an IPSL device
# VS-VIRTUAL-MACHINE: Launch a VM viewer session to a virtual machine.
#
deviceTypeProgrammaticName='ADX-IP-IQ'

# Filter device list by the device type programmatic name
filterIPIQDevices = '?filter=device_type_programmatic_name ' + deviceTypeProgrammaticName

# Store the JWT token to use in the HTTP request header
headers = {"Authorization": "Bearer " + jwt}

# Invoke REST API call to get the list of target devices in the DSView solution appliance and
# filter the devices by device type programmatic name
try:
    response = requests.get(Host + devicesURL + filterIPIQDevices,
                           timeout=TIMEOUT_SEC,
                           headers=headers,
                           verify=False)
except (requests.exceptions.ConnectTimeout, requests.exceptions.ReadTimeout):
    print('Unable to get list of devices from DSView solution appliance')
    exit()

# The target device ID
deviceId = ''
```

```
# Initialize the connection path type that is required to determine the viewer type for launching viewer sessions to
an IPUHD target device
connectionPathType=''

# Make sure the REST API call was successful
if response.status_code == 200:
    print('Getting the list of devices from the DSView solution appliance was successful')

    # Get the list of devices in the HTTP response
    json_response = response.json()
    deviceList = json_response['devices']
    print("The list of devices in the DSView solution appliance: ", json_response['devices'])

    # Make sure the device list has at least one item that matches the device type programmatic name
    print('device list length=', len(deviceList))
    if len(deviceList) == 0:
        print('There are no devices that match the device type programmatic name: ', deviceTypeProgrammaticName )
        exit()

    # Specify a target device from the device list (e.g., first target device or any other target)
    deviceInfo = deviceList[0]
    print('The information for the first device in the device list: ', deviceInfo)

    # Get the device name for the target device
    deviceName = deviceInfo['device_name']
    print('The device name for the target device: ', deviceName)

    # Get the device ID for the target device
    deviceId = deviceInfo['id']
    print('The device ID for the target device: ', deviceId)

    # Get the connection path type
    connectionPathType = deviceInfo['connection_path_info'][0]['path_type']
    print('The connection path type for the target device: ', connectionPathType)

else:
    print('Unable to get devices from DSView solution appliance. HTTP Response code is ', response.status_code)
    exit()

# For testing KVM or Serial viewer sessions to an IPUHD target device, specify the connection path type
connectionPathType='DCP_KVM_IP'

# Get the viewer type that is associated with a device programmatic name
print('Get the viewer type based on device type programmatic name: ', deviceTypeProgrammaticName)
viewerType = 'Unsupported'
if deviceTypeProgrammaticName == 'ADX-IP-IQ':
    viewerType = 'KVM'
elif deviceTypeProgrammaticName == 'ADX-IP-UHD' and connectionPathType == 'DCP_KVM_IP':
    viewerType = 'KVM'
elif deviceTypeProgrammaticName == 'ADX-IP-UHD' and connectionPathType == 'DCP_SERIAL_IP':
    viewerType = 'SERIAL'
```

```
elif deviceTypeProgrammaticName == 'SERIAL-TARGET' or deviceTypeProgrammaticName == 'ACS' or \
    deviceTypeProgrammaticName == 'ADX-IP-SERIAL-PORT':
    viewerType = 'SERIAL'
elif deviceTypeProgrammaticName == 'VS-VIRTUAL-MACHINE':
    viewerType = 'VM'
else:
    print("Unsupported device type programmatic name", deviceTypeProgrammaticName)
print('Viewer Type=', viewerType)

# STEP 3 - Create a viewer session ticket to launch a Viewer session to a target device
print('Create a viewer session ticket to launch a Viewer session to a target device')

# The REST endpoint to get the ticket ID for launching a viewer session to a target device
ticketURL = '/api/v1/userSessions/'

# Invoke REST API call to get the ticket ID for launching a viewer session
print('REST API Info to get ticket ID: ', Host + ticketURL + sessionId + ':createTicket')
try:
    response = requests.post(Host + ticketURL + sessionId + ':createTicket',
                             timeout=TIMEOUT_SEC,
                             headers=headers,
                             verify=False)
except (requests.exceptions.ConnectTimeout, requests.exceptions.ReadTimeout):
    print('Unable to get ticket information for launching Viewer sessions.')
    exit()

# The ticket ID that will be used for launching viewer session
ticketId = ''

# Make sure the REST API call was successful
if response.status_code == 200:
    print('Getting the ticket ID for launching a viewer session was successful.')

    # Get the ticket ID from the HTTP response
    ticketResponse = response.json()
    ticketId = ticketResponse['ticket']
else:
    print('Unable to get ticket ID. HTTP Response code is ', response.status_code)
    exit()
```

```
# Display the ticket ID
print('ticket ID=', ticketId)

# STEP 4 - Create URL to launch a viewer session to a target device

# Build URL to launch viewer session in a web browser
print('Create URL to launch a viewer session to a target device')
viewerURL = Host + '/api/v1/launchUI/?launchType=viewer&viewerType=' + viewerType + '&deviceId=' + deviceId +
 '&ticket=' + ticketId

# (Optional) - Add Sharing mode (i.e., STANDALONE_PASSIVE) to the URL
viewerURL += '&sharingMode=STANDALONE_PASSIVE'

# (Optional) - Add Mouse mode (i.e., 0 for absolute mode) to the URL
viewerURL += '&mouseMode=0'

# STEP 5 - Launch the viewer session in the default browser
print('Viewer Session URL=', viewerURL)
webbrowser.open(viewerURL, new=1)
```