```python
import tkinter as tk
from tkinter import ttk, messagebox
from onvif import ONVIFCamera
import threading
from PIL import Image, ImageTk
import requests
from io import BytesIO
from requests.auth import HTTPBasicAuth, HTTPDigestAuth
import time
import os


class OnvifCameraApp:
    def __init__(self, root):
        self.root = root
        self.root.title("ONVIF Camera Information")
        self.root.geometry("700x650")  # Increased height for
new components
        self.root.resizable(True, True)

        # Create main frame
        main_frame = ttk.Frame(root, padding="10")
        main_frame.pack(fill=tk.BOTH, expand=True)

        # Input frame
        input_frame = ttk.LabelFrame(main_frame, text="Camera
Connection", padding="10")
        input_frame.pack(fill=tk.X, padx=5, pady=5)

        # IP Address
        ttk.Label(input_frame, text="IP Address:").grid(row=0,
column=0, sticky=tk.W, pady=5)
        self.ip_var = tk.StringVar(value="192.168.202.169")
        ttk.Entry(input_frame, textvariable=self.ip_var,
width=20).grid(row=0, column=1, sticky=tk.W, padx=5)

        # Port
        ttk.Label(input_frame, text="Port:").grid(row=0,
column=2, sticky=tk.W, pady=5, padx=(10, 0))
        self.port_var = tk.StringVar(value="80")
        ttk.Entry(input_frame, textvariable=self.port_var,
width=10).grid(row=0, column=3, sticky=tk.W, padx=5)
```

```
        # Username
        ttk.Label(input_frame, text="Username:").grid(row=1,
column=0, sticky=tk.W, pady=5)
        self.username_var = tk.StringVar(value="root")
        ttk.Entry(input_frame, textvariable=self.username_var,
width=20).grid(row=1, column=1, sticky=tk.W, padx=5)

        # Password
        ttk.Label(input_frame, text="Password:").grid(row=1,
column=2, sticky=tk.W, pady=5, padx=(10, 0))
        self.password_var = tk.StringVar()
        ttk.Entry(input_frame, textvariable=self.password_var,
width=20, show="*").grid(row=1, column=3, sticky=tk.W,

padx=5)

        # Connection button frame
        button_frame = ttk.Frame(main_frame)
        button_frame.pack(fill=tk.X, padx=5, pady=5)

        # Connect button
        self.connect_btn = ttk.Button(button_frame,
text="Connect to Camera", command=self.on_connect_click)
        self.connect_btn.pack(side=tk.LEFT, padx=5)

        # Capture snapshot button
        self.capture_btn = ttk.Button(button_frame,
text="Capture Image", command=self.capture_image)
        self.capture_btn.pack(side=tk.RIGHT, padx=5)

        # User management frame
        user_management_frame = ttk.LabelFrame(main_frame,
text="ONVIF User Management", padding="10")
        user_management_frame.pack(fill=tk.X, padx=5, pady=5)

        # Username for adding
        ttk.Label(user_management_frame, text="New
Username:").grid(row=0, column=0, sticky=tk.W, pady=5)
        self.new_user_var = tk.StringVar()
        ttk.Entry(user_management_frame,
textvariable=self.new_user_var, width=20).grid(row=0,
column=1, padx=5)
```

```
        # Password for adding
        ttk.Label(user_management_frame, text="New
Password:").grid(row=0, column=2, sticky=tk.W, pady=5)
        self.new_password_var = tk.StringVar()
        ttk.Entry(user_management_frame,
textvariable=self.new_password_var, width=20,
show="*").grid(row=0, column=3,

padx=5)

        # Add User Button
        self.add_user_btn = ttk.Button(user_management_frame,
text="Add ONVIF User", command=self.add_onvif_user)
        self.add_user_btn.grid(row=0, column=4, padx=5)

        # List Users Button
        self.list_users_btn = ttk.Button(user_management_frame,
text="Refresh User List",
                                         command=self.refresh_user_list)
        self.list_users_btn.grid(row=1, column=4, padx=5,
pady=5)

        # Regular User Management Frame (New)
        regular_user_frame = ttk.LabelFrame(main_frame,
text="Regular User Management", padding="10")
        regular_user_frame.pack(fill=tk.X, padx=5, pady=5)

        # Admin Credentials Section
        admin_cred_frame = ttk.Frame(regular_user_frame)
        admin_cred_frame.grid(row=0, column=0, columnspan=5,
sticky=tk.W, pady=5)

        ttk.Label(admin_cred_frame, text="Admin credentials
required for user management:").grid(row=0, column=0,

columnspan=4,

sticky=tk.W)

        # Admin Username
        ttk.Label(admin_cred_frame, text="Admin
Username:").grid(row=1, column=0, sticky=tk.W, pady=5)
        self.admin_username_var = tk.StringVar()
```

```
        ttk.Entry(admin_cred_frame,
textvariable=self.admin_username_var, width=20).grid(row=1,
column=1, padx=5)

        # Admin Password
        ttk.Label(admin_cred_frame, text="Admin
Password:").grid(row=1, column=2, sticky=tk.W, pady=5,
padx=(10, 0))
        self.admin_password_var = tk.StringVar()
        ttk.Entry(admin_cred_frame,
textvariable=self.admin_password_var, width=20,
show="*").grid(row=1, column=3,

padx=5)

        # Separator
        ttk.Separator(regular_user_frame,
orient='horizontal').grid(row=1, column=0, columnspan=5,
sticky='ew', pady=10)

        # Regular User Info
        ttk.Label(regular_user_frame, text="New Regular
Username:").grid(row=2, column=0, sticky=tk.W, pady=5)
        self.reg_username_var = tk.StringVar()
        ttk.Entry(regular_user_frame,
textvariable=self.reg_username_var, width=20).grid(row=2,
column=1, padx=5)

        ttk.Label(regular_user_frame, text="New Regular
Password:").grid(row=2, column=2, sticky=tk.W, pady=5)
        self.reg_password_var = tk.StringVar()
        ttk.Entry(regular_user_frame,
textvariable=self.reg_password_var, width=20,
show="*").grid(row=2, column=3,

padx=5)

        # User Role Dropdown
        ttk.Label(regular_user_frame, text="User
Role:").grid(row=3, column=0, sticky=tk.W, pady=5)
        self.user_role_var = tk.StringVar(value="User")
        role_combobox = ttk.Combobox(regular_user_frame,
```

```
textvariable=self.user_role_var, width=15)
        role_combobox['values'] = ('Administrator', 'Operator',
'User', 'Anonymous')
        role_combobox.grid(row=3, column=1, padx=5,
sticky=tk.W)
        role_combobox.current(2)  # Default to "User"

        # Add Regular User Button
        self.add_reg_user_btn = ttk.Button(regular_user_frame,
text="Add Regular User", command=self.add_regular_user)
        self.add_reg_user_btn.grid(row=3, column=3, padx=5,
pady=5)

        # Results area
        result_frame = ttk.LabelFrame(main_frame, text="Camera
Information", padding="10")
        result_frame.pack(fill=tk.BOTH, expand=True, padx=5,
pady=5)

        # Scrolled text widget for results
        self.result_text = tk.Text(result_frame, wrap=tk.WORD,
height=15)
        self.result_text.pack(fill=tk.BOTH, expand=True, padx=5,
pady=5)
        self.result_text.config(state=tk.DISABLED)

        # Add scrollbar to the text widget
        scrollbar = ttk.Scrollbar(self.result_text, orient="vertical",
command=self.result_text.yview)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.result_text.config(yscrollcommand=scrollbar.set)

        # Image display area
        self.image_label = ttk.Label(result_frame)
        self.image_label.pack(pady=10)

        # Status bar
        self.status_var = tk.StringVar(value="Ready")
        status_bar = ttk.Label(root, textvariable=self.status_var,
relief=tk.SUNKEN, anchor=tk.W)
        status_bar.pack(side=tk.BOTTOM, fill=tk.X)

        # Initialize camera object
```

```
        self.camera = None
        self.has_admin_access = False

    def on_connect_click(self):
        # Start the camera connection in a background thread
        self.connect_thread =
threading.Thread(target=self.connect_to_camera)
        self.connect_thread.start()

    def connect_to_camera(self):
        try:
            # Update status
            self.status_var.set("Connecting to camera...")

            # Get camera details from user inputs
            ip = self.ip_var.get()
            port = self.port_var.get()
            username = self.username_var.get()
            password = self.password_var.get()

            # Initialize camera connection - removed wsdl_dir
parameter to use default
            self.camera = ONVIFCamera(ip, port, username,
password)
            time.sleep(1)  # Brief pause to ensure connection is
established

            # Get camera system information
            info = self.get_camera_info(self.camera)

            # Get user information in a structured way
            users_info = self.get_camera_users(self.camera)

            # Check if the connected user has admin rights
            self.check_admin_access()

            # Combine the information
            complete_info = info + "\n\n============== Users
==============\n" + users_info

            # If admin access is verified, note it
            if self.has_admin_access:
                complete_info += "\n\nAdmin access verified for
```

```
current user."

            # Update UI
            self.status_var.set(f"Connected to {ip}")
            self.update_result_text(complete_info)

        except Exception as e:
            error_msg = f"Error: {str(e)}"
            self.status_var.set(error_msg)
            self.update_result_text(error_msg)

    def check_admin_access(self):
        """Check if the current user has admin access rights"""
        try:
            # Try to perform an admin-only operation to test rights
            devicemgmt =
self.camera.create_devicemgmt_service()

            # Get user information - if this succeeds, likely has
admin rights
            users = devicemgmt.GetUsers()

            # Test succeeded, mark as having admin access
            self.has_admin_access = True

            # If admin fields are empty, pre-fill with current
credentials
            if not self.admin_username_var.get():

self.admin_username_var.set(self.username_var.get())
            if not self.admin_password_var.get():
                self.admin_password_var.set(self.password_var.get())

        except Exception as e:
            # If operation fails, user likely doesn't have admin
rights
            self.has_admin_access = False
            print(f"User doesn't appear to have admin rights:
{str(e)}")

    def get_camera_info(self, camera):
        # Access the devicemgmt service to get device info
        devicemgmt = camera.create_devicemgmt_service()
```

```
        # Get device information
        system_info = devicemgmt.GetDeviceInformation()

        # Collect the relevant camera details
        camera_details = (
            f"Manufacturer: {system_info.Manufacturer}\n"
            f"Model: {system_info.Model}\n"
            f"Firmware Version: {system_info.FirmwareVersion}\n"
            f"Serial Number: {system_info.SerialNumber}\n"
            f"Hardware ID: {system_info.HardwareId}\n"
        )

        # Get network info if available
        try:
            network_interfaces =
devicemgmt.GetNetworkInterfaces()
            if network_interfaces:
                camera_details += "\n============== Network
==============\n"
                for interface in network_interfaces:
                    if hasattr(interface, 'Info') and
hasattr(interface.Info, 'Name'):
                        camera_details += f"Interface:
{interface.Info.Name}\n"
                    if hasattr(interface, 'IPv4'):
                        if hasattr(interface.IPv4, 'Config') and
hasattr(interface.IPv4.Config, 'Manual'):
                            for addr in interface.IPv4.Config.Manual:
                                camera_details += f"IP:
{addr.Address}/{addr.PrefixLength}\n"
        except Exception as e:
            camera_details += f"\nError fetching network info:
{str(e)}\n"

        return camera_details

    def get_camera_users(self, camera):
        # Try multiple approaches to get user information
        users_info = ""

        # Approach 1: Using device management service
        try:
```

```
            devicemgmt = camera.create_devicemgmt_service()
            users = devicemgmt.GetUsers()

            if users:
                users_info += "--- Device Management Users ---\n"
                for user in users:
                    user_info = f"Username: {user.Username}, "
                    if hasattr(user, 'UserLevel'):
                        user_info += f"Level: {user.UserLevel}"
                    elif hasattr(user, 'Role'):
                        user_info += f"Role: {user.Role}"
                    users_info += user_info + "\n"
        except Exception as e:
            users_info += f"Failed to get users via Device
Management: {str(e)}\n"

        # Approach 2: Using device security service if available
        try:
            # Try to create security service
            device_service = camera.devicemgmt
            security_path =
device_service.GetServices({'IncludeCapability': False})

            security_service = None
            for service in security_path:
                if 'Security' in service.Namespace:
                    security_service =
camera.create_service('Security', service.XAddr)
                    break

            if security_service:
                users = security_service.GetUsers()

                if users:
                    users_info += "\n--- Security Service Users ---\n"
                    for user in users:
                        user_info = f"Username: {user.Username}, "
                        if hasattr(user, 'UserLevel'):
                            user_info += f"Level: {user.UserLevel}"
                        elif hasattr(user, 'Role'):
                            user_info += f"Role: {user.Role}"
                        users_info += user_info + "\n"
```

```
        except Exception as e:
            users_info += f"Failed to get users via Security Service: {str(e)}\n"

        # Approach 3: Direct access attempt for specialized cameras
        try:
            # Get device information and capabilities
            devicemgmt = camera.create_devicemgmt_service()
            capabilities = devicemgmt.GetCapabilities()

            # Check if the device has a user service endpoint
            if hasattr(capabilities, 'Security') and hasattr(capabilities.Security, 'UserManagement'):
                if capabilities.Security.UserManagement:
                    users_info += "\n--- User Management Capability: Available ---\n"

                    # Get all user levels if available
                    try:
                        user_levels = devicemgmt.GetUserLevel()
                        users_info += f"User Levels: {', '.join(user_levels)}\n"
                    except:
                        pass

                    # Try to list available user management functions
                    try:
                        service_caps = devicemgmt.GetServiceCapabilities()
                        if hasattr(service_caps, 'Security'):
                            users_info += f"Security Capabilities: {service_caps.Security}\n"
                    except:
                        pass
        except Exception as e:
            users_info += f"Failed to get user capabilities: {str(e)}\n"

        # If all methods failed or returned no users
        if not users_info or ("Failed" in users_info and "Device Management Users" not in users_info):
```

```
            users_info += "\nNo user accounts could be retrieved. The camera may not support user management via ONVIF."

        return users_info

    def refresh_user_list(self):
        if not self.camera:
            self.status_var.set("Error: Connect to camera first")
            return

        try:
            # Get current text
            current_text = self.result_text.get(1.0, tk.END)

            # Find and remove old user section
            if "============== Users ==============" in current_text:
                basic_info = current_text.split("============== Users ==============")[0]
            else:
                basic_info = current_text

            # Get fresh user info
            users_info = self.get_camera_users(self.camera)

            # Update display
            complete_info = basic_info + "\n\n============== Users ==============\n" + users_info
            self.update_result_text(complete_info)
            self.status_var.set("User list refreshed")
        except Exception as e:
            self.status_var.set(f"Error refreshing users: {str(e)}")

    def add_onvif_user(self):
        if not self.camera:
            self.status_var.set("Error: Connect to camera first")
            return

        try:
            # Get user details
            username = self.new_user_var.get()
            password = self.new_password_var.get()
```

```
        if not username or not password:
            self.status_var.set("Error: Username and password
required")
            return

        # Try multiple methods to add a user

        # Method 1: Using Device Management service
        try:
            devicemgmt =
self.camera.create_devicemgmt_service()

            # Prepare user data (adjust User role/level as
needed)
            user_data = {
                'Username': username,
                'Password': password,
                'UserLevel': 'User'  # Could be 'Administrator',
'Operator', 'User', etc.
            }

            # Create the user
            devicemgmt.CreateUsers(user_data)
            self.status_var.set(f"User {username} added
successfully")

            # Refresh user list
            self.refresh_user_list()
            return
        except Exception as e:
            print(f"Method 1 failed: {str(e)}")

        # Method 2: Using Security service if available
        try:
            # Try to find and create security service
            device_service = self.camera.devicemgmt
            security_path =
device_service.GetServices({'IncludeCapability': False})

            security_service = None
            for service in security_path:
                if 'Security' in service.Namespace:
                    security_service =
```

```
self.camera.create_service('Security', service.XAddr)
                    break

            if security_service:
                # Prepare user data
                user_data = {
                    'Username': username,
                    'Password': password,
                    'Role': 'User'  # Could be 'Administrator',
'Operator', 'User', etc.
                }

                # Create the user
                security_service.CreateUser(user_data)
                self.status_var.set(f"User {username} added
successfully")

                # Refresh user list
                self.refresh_user_list()
                return
            else:
                raise Exception("Security service not available")
        except Exception as e:
            print(f"Method 2 failed: {str(e)}")

        # If we got here, both methods failed
        raise Exception("Failed to add user with any available
method")

    except Exception as e:
        error_msg = f"Error adding user: {str(e)}"
        self.status_var.set(error_msg)
        self.update_result_text(f"{self.result_text.get(1.0,
tk.END)}\n\n{error_msg}")

def add_regular_user(self):
    """Add a regular (non-ONVIF) user to the camera using
HTTP API"""
    if not self.camera:
        self.status_var.set("Error: Connect to camera first")
        return

    # Get user details
```

```
        username = self.reg_username_var.get()
        password = self.reg_password_var.get()
        role = self.user_role_var.get()

        # Get admin credentials
        admin_username = self.admin_username_var.get()
        admin_password = self.admin_password_var.get()

        if not username or not password:
            self.status_var.set("Error: Username and password
required")
            return

        if not admin_username or not admin_password:
            self.status_var.set("Error: Admin credentials required for
user management")
            return

        # Update status
        self.status_var.set(f"Adding regular user {username}...")

        try:
            # Get camera IP and port
            ip = self.ip_var.get()
            port = self.port_var.get()

            # Try to add user using multiple methods
            self.add_user_multi_method(ip, port, admin_username,
admin_password, username, password, role)

        except Exception as e:
            error_msg = f"Error adding regular user: {str(e)}"
            self.status_var.set(error_msg)
            self.update_result_text(f"{self.result_text.get(1.0,
tk.END)}\n\n{error_msg}")

    def add_user_multi_method(self, ip, port, admin_username,
admin_password, new_username, new_password, role):
        """Try multiple methods to add a regular user to the
camera"""

        # Method 1: Try using CGI API (common in many cameras)
        try:
```

```
            # Build URL for user creation (varies by camera
make/model)
            url = f"http://{ip}:{port}/cgi-bin/admin/userconfig.cgi"

            # Parameters for user creation
            params = {
                'action': 'add',
                'user': new_username,
                'password': new_password,
                'group': role.lower()  # Convert to lowercase as many
cameras expect
            }

            # Make request with admin authentication
            response = requests.get(
                url,
                params=params,
                auth=HTTPBasicAuth(admin_username,
admin_password),
                timeout=10
            )

            if response.status_code == 200 and ('success' in
response.text.lower() or 'ok' in response.text.lower()):
                self.status_var.set(f"Regular user {new_username}
added successfully")
                self.refresh_user_list()
                return True

        except Exception as e:
            print(f"CGI method failed: {str(e)}")

        # Method 2: Try using API endpoint (common in newer
cameras)
        try:
            url = f"http://{ip}:{port}/api/users"

            # JSON payload for user creation
            payload = {
                'username': new_username,
                'password': new_password,
                'role': role,
                'enabled': True
```

```
        }

        # Make POST request with admin authentication
        response = requests.post(
            url,
            json=payload,
            auth=HTTPBasicAuth(admin_username,
admin_password),
            timeout=10
        )

        if response.status_code in [200, 201]:
            self.status_var.set(f"Regular user {new_username}
added successfully")
            self.refresh_user_list()
            return True

    except Exception as e:
        print(f"REST API method failed: {str(e)}")

    # Method 3: Try alternative CGI path (common in some
Asian cameras)
    try:
        url = f"http://{ip}:{port}/stw-cgi/system/users.cgi"

        # Form data for user creation
        data = {
            'action': 'add',
            'userName': new_username,
            'password': new_password,
            'userType': self.map_role_to_type(role),
            'enable': '1'
        }

        # Try both auth methods, as some cameras use Digest
        try:
            response = requests.post(
                url,
                data=data,
                auth=HTTPBasicAuth(admin_username,
admin_password),
                timeout=10
            )
```

```
            except:
                response = requests.post(
                    url,
                    data=data,
                    auth=HTTPDigestAuth(admin_username,
admin_password),
                    timeout=10
                )

            if response.status_code == 200:
                self.status_var.set(f"Regular user {new_username}
added successfully")
                self.refresh_user_list()
                return True

        except Exception as e:
            print(f"Alternative CGI method failed: {str(e)}")

        # Method 4: Try using ONVIF "CreateUsers" with
temporary admin login
        try:
            # Create temporary camera connection with admin
credentials
            admin_camera = ONVIFCamera(ip, port,
admin_username, admin_password)
            time.sleep(1)  # Brief pause to ensure connection

            # Create device management service
            devicemgmt =
admin_camera.create_devicemgmt_service()

            # Prepare user data
            user_data = {
                'Username': new_username,
                'Password': new_password,
                'UserLevel': self.map_role_to_onvif_level(role)
            }

            # Create the user
            devicemgmt.CreateUsers(user_data)
            self.status_var.set(f"Regular user {new_username}
added successfully via ONVIF")
            self.refresh_user_list()
```

```
            return True

        except Exception as e:
            print(f"ONVIF admin method failed: {str(e)}")

        # If all methods failed, raise an exception
        raise Exception(
            "Failed to add regular user with any available method. The camera may not support this operation or the admin credentials may be incorrect.")

    def map_role_to_type(self, role):
        """Map standard role names to camera-specific user types"""
        role_map = {
            'Administrator': '1',
            'Admin': '1',
            'Operator': '2',
            'User': '3',
            'Anonymous': '4'
        }
        return role_map.get(role, '3')  # Default to User level

    def map_role_to_onvif_level(self, role):
        """Map UI roles to ONVIF UserLevel values"""
        role_map = {
            'Administrator': 'Administrator',
            'Admin': 'Administrator',
            'Operator': 'Operator',
            'User': 'User',
            'Anonymous': 'Anonymous'
        }
        return role_map.get(role, 'User')  # Default to User level

    def capture_image(self):
        if not self.camera:
            self.status_var.set("Error: Connect to camera first")
            return

        # Fetch the snapshot from the camera using the ONVIF camera methods
        try:
            # Update status
```

```
            self.status_var.set("Capturing image...")

            # Get media service and profile
            media = self.camera.create_media_service()
            profiles = media.GetProfiles()

            if not profiles:
                raise Exception("No media profiles found")

            profile_token = profiles[0].token

            # Capture a snapshot
            media_snapshot =
media.GetSnapshotUri({'ProfileToken': profile_token})
            snapshot_url = media_snapshot.Uri

            # Attempt to fetch the snapshot with authentication
            response = self.fetch_image_with_auth(snapshot_url)

            # Check for a successful response (HTTP status 200)
            if response.status_code == 200:
                # Try to open the image
                image = Image.open(BytesIO(response.content))

                # Display the image in the tkinter window
                self.display_image(image)
                self.status_var.set("Image captured successfully")
            else:
                raise Exception(f"Failed to fetch image: HTTP status
code {response.status_code}")

        except Exception as e:
            error_msg = f"Error capturing image: {str(e)}"
            self.status_var.set(error_msg)
            self.update_result_text(f"{self.result_text.get(1.0,
tk.END)}\n\n{error_msg}")

    def fetch_image_with_auth(self, snapshot_url):
        # Attempt Basic Authentication
        try:
            response = requests.get(snapshot_url,
auth=HTTPBasicAuth(self.username_var.get(),
self.password_var.get()),
```

```
                                timeout=5)
            if response.status_code == 200:
                return response
        except Exception as e:
            print(f"Basic Authentication failed: {str(e)}")

        # If Basic Authentication failed, attempt Digest
Authentication
        try:
            response = requests.get(snapshot_url,
auth=HTTPDigestAuth(self.username_var.get(),
self.password_var.get()),
                                timeout=5)
            if response.status_code == 200:
                return response
        except Exception as e:
            print(f"Digest Authentication failed: {str(e)}")

        # Try without authentication as a last resort
        try:
            response = requests.get(snapshot_url, timeout=5)
            if response.status_code == 200:
                return response
        except Exception as e:
            print(f"No authentication attempt failed: {str(e)}")

        # If all methods fail, raise an error
        raise Exception("All authentication methods failed for
fetching the snapshot.")

    def display_image(self, image):
        # Resize and display the image in the tkinter window
        image.thumbnail((400, 400))
        photo = ImageTk.PhotoImage(image)

        # Update the image in the label
        self.image_label.config(image=photo)
        self.image_label.image = photo

    def update_result_text(self, text):
        # Use root.after to ensure this runs in the main thread
        self.root.after(0, self._update_result_text, text)
```

```
    def _update_result_text(self, text):
        # This method actually updates the Text widget in the
main thread
        self.result_text.config(state=tk.NORMAL)
        self.result_text.delete(1.0, tk.END)
        self.result_text.insert(tk.END, text)
        self.result_text.config(state=tk.DISABLED)

def main():
    root = tk.Tk()
    app = OnvifCameraApp(root)
    root.mainloop()

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