```
from onvif import ONVIFCamera
import time
import os
import csv

# Camera-specific credentials [MOST SAFE CODING PRACTICES,
ONLY THE MOST SAFE WITH CUOPS]
camera_credentials = {
    "Axis": {"username": "testuser", "password":
"0rc#1dVMS2023"},
    "Vivotek": {"username": "root", "password":
"0rcH1dVMS2023"},
    "Hanwha": {"username": "admin", "password":
"0rc#1dVMS2023"},
    "I-Pro TIGER": {"username": "admin", "password":
"0rc#1dVMS2023"},
    "Pelco": {"username": "root", "password":
"0rc#1dVMS2023"},
    "Bosch": {"username": "service", "password":
"0rc#1dVMS2023"},
    "Avigilon": {"username": "administrator", "password":
"tpain"},
    "IC Realtime": {"username": "admin", "password":
"0rc#1dVMS2023"}
}


# Read IPs and MACs from camera brand files
def read_camera_ips_and_macs(filename):
    camera_ips_and_macs = []
    try:
        with open(filename, "r") as file:
            lines = file.readlines()
            for line in lines[1:]:  # Skip header
                parts = line.split("\t")
                if len(parts) >= 2:
                    ip = parts[0].strip()
                    mac = parts[1].strip()  # Get MAC address from
first script file
                    camera_ips_and_macs.append((ip, mac))
    except Exception as e:
        print(f"Error reading {filename}: {e}")
```

```
    return camera_ips_and_macs

# Function to get supported codecs from a camera
def get_codec_support(camera):
    try:
        # Create media service
        media_service = camera.create_media_service()

        # Get video encoder configuration options
        request =
media_service.create_type('GetVideoEncoderConfigurationOpti
ons')

        # Try to get configurations - some cameras require a
profile token
        try:
            # First try without specific profile token
            encoder_options =
media_service.GetVideoEncoderConfigurationOptions(request)
        except:
            # If that fails, get the first profile and try with its token
            profiles = media_service.GetProfiles()
            if profiles:
                request.ConfigurationToken =
profiles[0].VideoEncoderConfiguration.token
                encoder_options =
media_service.GetVideoEncoderConfigurationOptions(request)
            else:
                return "No profiles found"

        # Extract supported codecs
        supported_codecs = []

        # Check if H.264 is supported
        if hasattr(encoder_options, 'H264'):
            supported_codecs.append('H264')

        # Check if H.265 is supported
        if hasattr(encoder_options, 'H265'):
            supported_codecs.append('H265')

        # Check if MJPEG is supported
```

```
        if hasattr(encoder_options, 'JPEG'):
            supported_codecs.append('MJPEG')

        # Get current codec (from first profile)
        current_codec = "Unknown"
        try:
            profiles = media_service.GetProfiles()
            if profiles:
                video_config =
media_service.GetVideoEncoderConfiguration(
                    {'ConfigurationToken':
profiles[0].VideoEncoderConfiguration.token}
                )
                current_codec = video_config.Encoding
        except:
            pass

        return {
            'supported_codecs': ','.join(supported_codecs) if
supported_codecs else "Unknown",
            'current_codec': current_codec
        }
    except Exception as e:
        print(f"Error getting codec support: {e}")
        return {
            'supported_codecs': f"Error: {str(e)}",
            'current_codec': "Error"
        }
# Function to check if the camera has PTZ capabilities
def check_ptz_capabilities(camera):
    try:
        # Create PTZ service
        ptz_service = camera.create_ptz_service()

        # Try to get PTZ configurations - if it succeeds, the camera
has PTZ
        try:
            ptz_configs = ptz_service.GetConfigurations()
            if ptz_configs:
                return "Yes"
            else:
                return "No"  # No configurations found
```

```
        except:
            # Get device capabilities to check for PTZ support
            device_service = camera.create_devicemgmt_service()
            device_capabilities = device_service.GetCapabilities()

            if hasattr(device_capabilities, 'PTZ') and
device_capabilities.PTZ:
                return "Yes"
            else:
                return "No"
    except Exception as e:
        print(f"Error checking PTZ capabilities: {e}")
        return "Unknown"


# Function to check audio capabilities
def check_audio_capabilities(camera):
    try:
        # Create media service
        media_service = camera.create_media_service()

        # Check audio encoder configurations (for output)
        has_audio_output = False
        has_audio_input = False

        try:
            # Check for audio output (speaker)
            audio_outputs =
media_service.GetAudioOutputConfigurations()
            if audio_outputs and len(audio_outputs) > 0:
                has_audio_output = True
        except:
            pass

        try:
            # Check for audio input (microphone)
            audio_inputs =
media_service.GetAudioSourceConfigurations()
            if audio_inputs and len(audio_inputs) > 0:
                has_audio_input = True
        except:
            pass
```

```
        # Also try device capabilities for audio
        try:
            device_service = camera.create_devicemgmt_service()
            device_capabilities = device_service.GetCapabilities()

            if hasattr(device_capabilities, 'Media'):
                media_caps = device_capabilities.Media
                if hasattr(media_caps, 'AudioOutputs') and
media_caps.AudioOutputs:
                    has_audio_output = True
                if hasattr(media_caps, 'AudioSources') and
media_caps.AudioSources:
                    has_audio_input = True
        except:
            pass

        return {
            'has_microphone': 'Yes' if has_audio_input else 'No',
            'has_audio_output': 'Yes' if has_audio_output else 'No'
        }
    except Exception as e:
        print(f"Error checking audio capabilities: {e}")
        return {
            'has_microphone': 'Unknown',
            'has_audio_output': 'Unknown'
        }


# Function to check for analytics support (could indicate smart-
search)
def check_analytics_capabilities(camera):
    try:
        # Try to create the analytics service - if it fails, the camera
likely doesn't support analytics
        try:
            analytics_service = camera.create_analytics_service()

            # Try to get rules and modules
            rules = analytics_service.GetRules()
            modules = analytics_service.GetAnalyticsModules()

            # If we can get rules or modules, analytics is supported
            if rules or modules:
```

```
                return "Yes"

            # Check device capabilities for analytics
            device_service = camera.create_devicemgmt_service()
            capabilities = device_service.GetCapabilities()

            if hasattr(capabilities, 'Analytics') and
capabilities.Analytics:
                return "Yes"

            return "No"  # No analytics features found
        except:
            # Check device capabilities for analytics
            device_service = camera.create_devicemgmt_service()
            capabilities = device_service.GetCapabilities()

            if hasattr(capabilities, 'Analytics') and
capabilities.Analytics:
                return "Yes"
            else:
                return "No"
    except Exception as e:
        print(f"Error checking analytics capabilities: {e}")
        return "Unknown"


# Improved function to check for dewarp capabilities using
multiple approaches
def check_dewarp_capabilities(camera, manufacturer, model):
    try:
        dewarp_detected = False
        dewarp_reason = ""

        # 1. Check model name for fisheye indicators
        model_lower = model.lower() if model else ""

        fisheye_keywords = ["fisheye", "360", "180", "panoramic",
"pano", "dome"]
        for keyword in fisheye_keywords:
            if keyword in model_lower:
                dewarp_detected = True
                dewarp_reason = f"Model name contains
'{keyword}'"
```

```
            break

        # 2. Try ONVIF imaging service approach if not already
detected
        if not dewarp_detected:
            try:
                imaging_service = camera.create_imaging_service()

                # Get profiles to get video sources
                media_service = camera.create_media_service()
                profiles = media_service.GetProfiles()

                if profiles:
                    video_source_token =
profiles[0].VideoSourceConfiguration.SourceToken

                    # Try to get imaging settings options
                    options =
imaging_service.GetOptions({'VideoSourceToken':
video_source_token})

                    # Check for properties that might indicate dewarp
capability
                    if (hasattr(options, 'Dewarping') or
                            hasattr(options, 'DistortionCorrection') or
                            hasattr(options, 'LensDistortionCorrection')):
                        dewarp_detected = True
                        dewarp_reason = "ONVIF imaging settings"
            except:
                pass

        # 3. Manufacturer-specific checks as fallback
        if not dewarp_detected:
            # Check manufacturer for typical fisheye camera
makers
            manufacturer_lower = manufacturer.lower() if
manufacturer else ""

            # Manufacturer-specific model patterns
            if manufacturer_lower == "axis" and any(
                    x in model_lower for x in ["m3057", "m3058",
"m30", "p3807", "q3819", "p9106"]):
                dewarp_detected = True
```

```
                dewarp_reason = "Known Axis fisheye model"
            elif manufacturer_lower == "hanwha" and any(x in
model_lower for x in ["pnf", "xnf", "qnf"]):
                dewarp_detected = True
                dewarp_reason = "Known Hanwha fisheye model"
            elif manufacturer_lower == "vivotek" and any(x in
model_lower for x in ["fe", "fe8", "cc8", "cc9"]):
                dewarp_detected = True
                dewarp_reason = "Known Vivotek fisheye model"
            elif manufacturer_lower == "bosch" and any(x in
model_lower for x in ["flexidome", "7000"]):
                dewarp_detected = True
                dewarp_reason = "Known Bosch fisheye model"
            elif manufacturer_lower == "avigilon" and "h4a-do" in
model_lower:
                dewarp_detected = True
                dewarp_reason = "Known Avigilon fisheye model"

        # Return results as a string with reason if detected
        if dewarp_detected:
            return f"Yes ({dewarp_reason})"
        else:
            return "No"

    except Exception as e:
        print(f"Error checking dewarp capabilities: {e}")
        return "Unknown"


# Updated function to log in to a camera and get details
def get_camera_info(camera_ip, username, password):
    try:
        # ONVIF login to camera
        camera = ONVIFCamera(camera_ip, 80, username,
password)

        # Create the device management service (for device
details)
        device_mgmt_service =
camera.create_devicemgmt_service()

        # Retrieve camera system information
        system_info =
```

```
device_mgmt_service.GetDeviceInformation()

        # Retrieve camera details
        manufacturer = system_info.Manufacturer  # Get the
manufacturer name
        model = system_info.Model
        firmware_version = system_info.FirmwareVersion
        serial_number = system_info.SerialNumber  # Get serial
number (as fallback for MAC)

        # Get codec support information
        codec_info = get_codec_support(camera)

        # Check for PTZ capabilities
        ptz_capability = check_ptz_capabilities(camera)

        # Check for audio capabilities
        audio_info = check_audio_capabilities(camera)

        # Check for analytics support (smart search)
        analytics_support = check_analytics_capabilities(camera)

        # Check for dewarp capabilities - now passing model
name too
        dewarp_capability = check_dewarp_capabilities(camera,
manufacturer, model)

        # Return camera details with all capabilities
        return (
            camera_ip,
            manufacturer,
            model,
            firmware_version,
            serial_number,
            codec_info['supported_codecs'],
            codec_info['current_codec'],
            ptz_capability,
            audio_info['has_microphone'],
            audio_info['has_audio_output'],
            analytics_support,
            dewarp_capability
        )
```

```
    except Exception as e:
        # Capture and return error message for the faulty camera
        return (camera_ip, "-", "-", "-", str(e), "-", "-", "Unknown",
"Unknown", "Unknown", "Unknown", "Unknown")


# Write camera information to a brand-specific file using proper
CSV handling
def write_camera_info(brand, camera_details,
camera_ips_and_macs):
    try:
        # Create a file for each brand: AxisCameraInformation.txt,
VivotekCameraInformation.txt, etc.
        filename = f"{brand}CameraInformation.txt"
        with open(filename, "w", newline='') as file:
            # Create CSV writer to handle fields with commas
properly
            csv_writer = csv.writer(file,
quoting=csv.QUOTE_MINIMAL)

            # Write header
            csv_writer.writerow([
                "IP ADDR", "Manufacturer", "Model", "Firmware",
                "MAC/Serial Number", "Supported Codecs", "Current
Codec",
                "PTZ Capable", "Has Microphone", "Has Audio
Output",
                "Smart Search Support", "Dewarp Capable", "Notes"
            ])

            # Loop through camera details and write them, adding
the correct MAC address
            for (ip, mac), details in zip(camera_ips_and_macs,
camera_details):
                if details:
                    # Prepare row data
                    row = [
                        details[0].strip(),  # IP
                        details[1].strip(),  # Manufacturer
                        details[2].strip(),  # Model
                        details[3].strip(),  # Firmware
                        mac,  # MAC
```

```
                    details[5],  # Supported Codecs
                    details[6],  # Current Codec
                    details[7],  # PTZ Capable
                    details[8],  # Has Microphone
                    details[9],  # Has Audio Output
                    details[10],  # Smart Search Support
                    details[11]  # Dewarp Capable
                ]

                # Add notes if available
                if len(details[4].strip()) > 0 and details[4] != "-":
                    row.append(details[4].strip())

                # Write the row
                csv_writer.writerow(row)

        print(f"Camera information saved to {filename}.")
    except Exception as e:
        print(f"Error writing to {brand}CameraInformation.txt:
{e}")


# Main execution flow
def main():
    print("Starting Enhanced Axis-ONVIF Scraper V2.0")
    print("IPconfigure.2024-2025.Timmons")
    input("Press Enter to start the program...")

    # Loop through each camera brand's file and process them
individually
    for brand, credentials in camera_credentials.items():
        filename = f"{brand}Cameraips.txt"  # For example,
"AxisCameraips.txt"
        print(f"Reading IPs and MACs from {filename} for brand:
{brand}")
        camera_ips_and_macs =
read_camera_ips_and_macs(filename)

        # List to store camera details for the current brand
        camera_details = []

        # Loop through the IPs, use ONVIF to get info, and store
the results
```

```
        for ip, mac in camera_ips_and_macs:
            print(f"Getting info for {brand} camera: {ip}")
            details = get_camera_info(ip, credentials["username"],
credentials["password"])
            camera_details.append(details)
            time.sleep(1)  # Sleep to prevent overwhelming the
cameras

        # Write the details to the corresponding brand-specific file
        write_camera_info(brand, camera_details,
camera_ips_and_macs)


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