MWCapture SDK macOS Examples Documentation

Preface

Examples

Overview

SDK Version 3 is Magewell's latest SDK release, supporting Magewell Pro capture and USB Family capture devices. Hereinafter, "all devices" refers to the aforesaid devices.

SDK Version 3 provides both private interfaces and general interfaces based on AVFoundation, balancing performance, flexibility, and versatility. For scenarios with high performance requirements that need flexible audio and video data capture (such as capturing only the upper-left quarter of a frame, capturing data at a custom frame rate, or capturing audio data in non-LPCM format), private interfaces are recommended. For scenarios using multiple family of Magewell capture devices, applications developed with general interfaces will be compatible across all supported devices.

The interfaces provided by SDK Version 3 can be categorized into two main types: audio/video data capture interfaces and control interfaces. Data capture interfaces are further divided into private and general interfaces:

- Private capture interfaces: Pro Capture cards
- General capture interfaces: Pro Capture cards, and USB Capture devices
- Control interfaces: All are private interfaces with consistent functionality across different device family. For example, the MWGetVideoStatus interface for retrieving input video signal status works with all capture devices.

To help developers quickly integrate the Magewell SDK into their applications and shorten the development cycle, SDK v3 includes numerous sample projects covering various common usage scenarios. Developers can select appropriate samples for reference based on their specific requirements.

Supported Products

- Pro Capture Family
- USB Capture Family

Supported Operating Systems

- macOS Monterey 12.7.4 and later
- Compatible with driver v1.4.64 and later
- Compatible with both Intel and Apple Silicon chips
- Note: For macOS versions prior to Monterey 12.7.4, please contact us through Tickets system for additional information.

Version History

Version V3.4.0 (2025/11/19)

- Compatible with Mac drivers V1.4.64 and later, and runs on Macs powered by either Apple silicon or Intel silicon.

Version 3.3.1.16905 (2020/09/30)

- Added the GPU-accelerated video encoding module.
- Added the MP4 recording module and MP4 recording library.
- Added the Mp4Repair example, which is used to repair MP4 files corrupted by unexpected termination of recording.
- Optimized the SDKv3 API documentation.

Example Introduction

CmdTools Sample

All samples in this section are command-line based.

- **AudioCapture**: This sample demonstrates the general process of audio data capture using private interfaces, with detailed explanations of the call logic for various interfaces in audio capture. It serves as a reference for developers working on audio capture. Note that this sample is only compatible with Pro Capture cards.
- **CaptureByInput**: This sample demonstrates a workflow for capturing video data via the private interfaces, triggered by message notifications. Compared to the "CaptureByTimer" sample, the resolution and frame rate captured depend on the signal source. The interval of notifications from the capture card matches the frame interval of the input signal, resulting in a fixed capture frame rate and a more stable operation workflow.
- **CaptureByTimer**: This sample demonstrates another workflow for capturing video data via the private interfaces, using a timer to set capture times (video capture is performed at specified times). Thus, the capture frame rate is adjustable. Compared to the "CaptureByInput" sample, it is more suitable for scenarios where the capture frame rate differs from the input frame rate.
- **GetANCPacket**: This sample demonstrates the general process of obtaining ANC (Ancillary) data for a specified channel using private interfaces, which developers with such requirements can reference.
- **HDMIInfoFrame**: This sample demonstrates the general process of retrieving HDMI information frames using private interfaces, which developers with such requirements can reference.
- **InputSignal**: This sample demonstrates the general process of obtaining detailed format information about input signals using private interfaces, including status information for the specified channel, key video signal parameters such as resolution and color space, and critical audio signal information such as sampling rate and format. Developers are strongly advised to carefully review this sample.
- **InputSource**: This sample demonstrates the general process of obtaining the type and quantity of audio and video input sources using private interfaces, which developers with such requirements can reference.
- **InputSignalNotify**: This sample demonstrates the process of monitoring input signal changes using private interfaces.
- **ReadWriteEDID**: This sample implements two processes through private interfaces: 1. Exporting the EDID of a specified channel as a binary file; 2. Writing a specified EDID file to a target channel. This sample provides a reference for developers implementing EDID read/write functionality for devices.
- **USBDeviceDetect**: This sample demonstrates the detection of USB capture device hot-swapping events using private interfaces, which developers with such requirements can reference.
- **SetUSBCaptureFourcc**: This sample demonstrates the general process of reading and configuring USB video color space attributes using private interfaces.
- **Mp4Repair**: This sample demonstrates how to repair corrupted video files—specifically, those recorded with LibMWMp4 that became damaged due to improper termination of recording processes (e.g., caused by abnormal power outages).

GUI Sample

All samples in this section are GUI-based.

- **AVCapture**: The main features of this sample include: (1) Using private interfaces to capture and render video; (2) Using private interfaces to capture and monitor audio; (3) Configuring video capture resolution, frame rate, and color space; (4) Multi-channel capture; (5) Recording captured audio and video data as FLV files. Compared to the "AVCapture2" sample, this sample uses private interfaces for both audio and video capture, resulting in higher capture efficiency and more abundant configurable capture parameters. However, this limits compatibility to only Pro family capture cards.
- **AVCapture2**: The main features of this sample include: (1) Using general interfaces to capture and render video; (2) Using general interfaces to capture and monitor audio; (3) Configuring video capture resolution, frame rate, and color space; (4) Multi-channel capture. Unlike "AVCapture", this sample internally leverages AVFoundation for capture implementation, enabling support for all Magewell capture devices with enhanced versatility. Additionally, you can capture audio and video from different devices.

AudioCapture

Supported Hardware Devices: Pro Capture cards

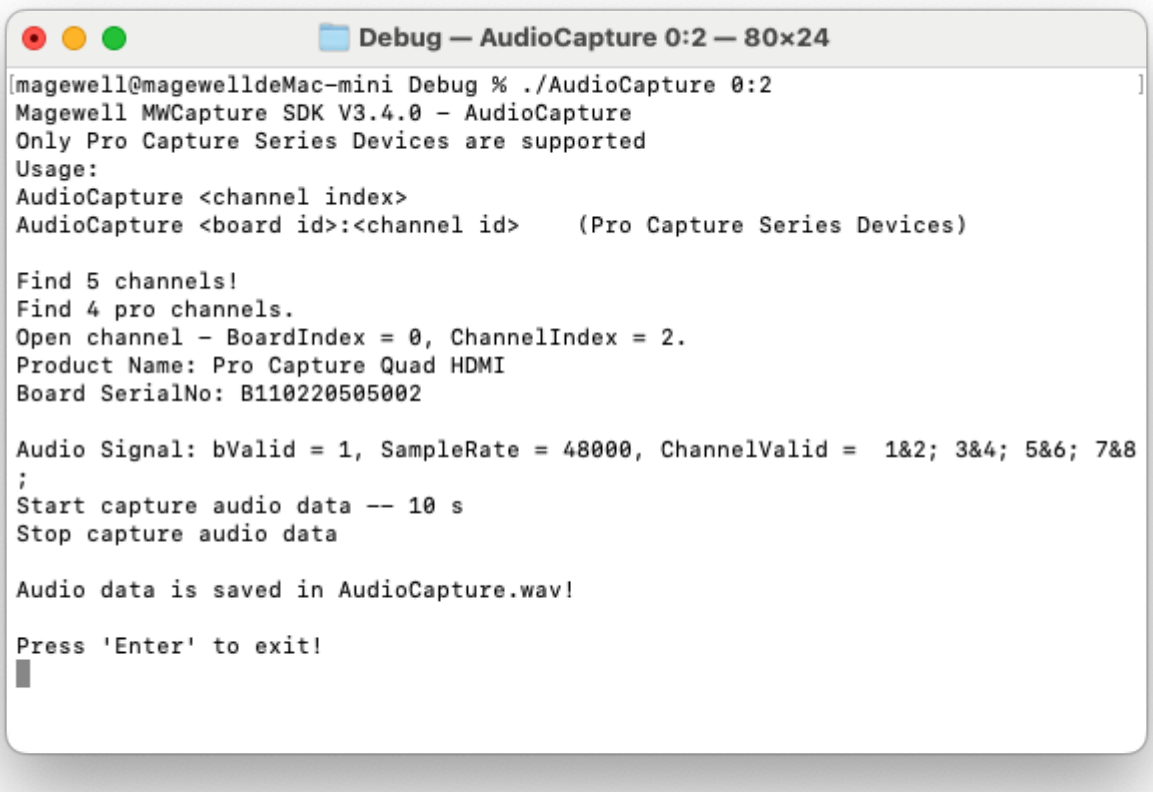
Contents Demonstrated in the Sample:

1. Capturing audio data from input signals
2. Saving LPCM format audio signals as WAV files
3. Channel selection via command parameters when multiple devices are available

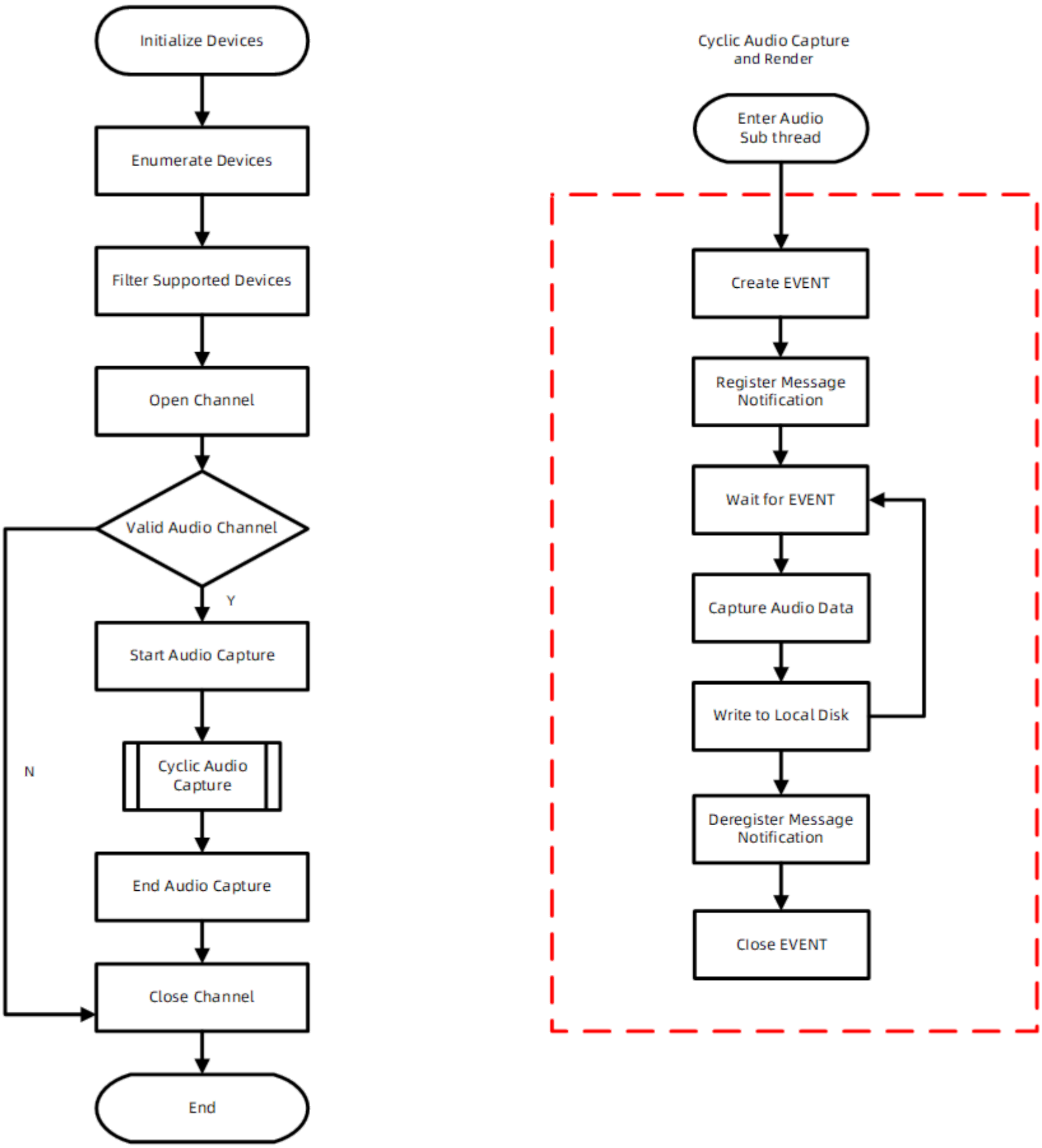
Call Logic:

1. Obtain version information, initialize, and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Filter supported devices based on device information: MWGetChannelInfoByIndex
3. Open the device:
    i. Open the device specified by command parameters: MWOpenChannel
    ii. When no device is specified in command parameters, open the first available channel by default: MWGetDevicePath, MWOpenChannelByPath
4. Retrieve audio source count and status: MWGetAudioInputSourceArray, MWGetAudioSignalStatus
5. Start audio capture: MWStartAudioCapture
6. Create event: MWCreateEvent
7. Register for message notifications: MWRegisterNotify
8. Wait for event: MWTryWaitEvent, MWWaitEvent
9. Capture one frame of audio data: MWCaptureAudioFrame
10. Write audio data to WAV file
11. Repeat steps 8, 9, and 10 until capture is stopped
12. Stop audio capture and release resources: MWUnregisterNotify, MWCloseEvent, MWCloseChannel, MWCaptureExitInstance

Result:



Flowchart:


Initialize Devices
Enumerate Devices
Filter Supported Devices
Open Channel
Valid Audio Channel
Y
N
Start Audio Capture
Cyclic Audio Capture
End Audio Capture
Close Channel
End
Cyclic Audio Capture and Render
Enter Audio Sub thread
Create EVENT
Register Message Notification
Wait for EVENT
Capture Audio Data
Write to Local Disk
Deregister Message Notification
Close EVENT

CaptureByInput

Supported Hardware Devices: Pro Capture cards

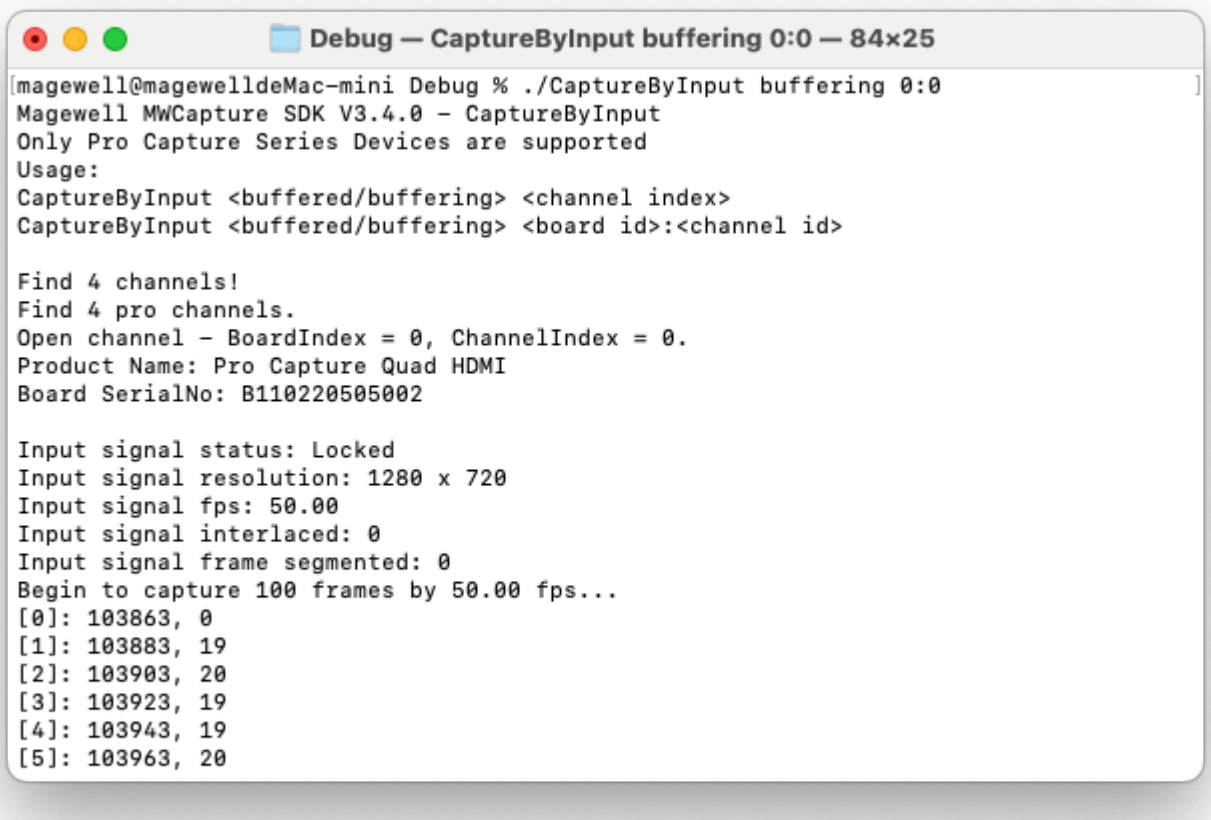
Contents Demonstrated in the Sample:

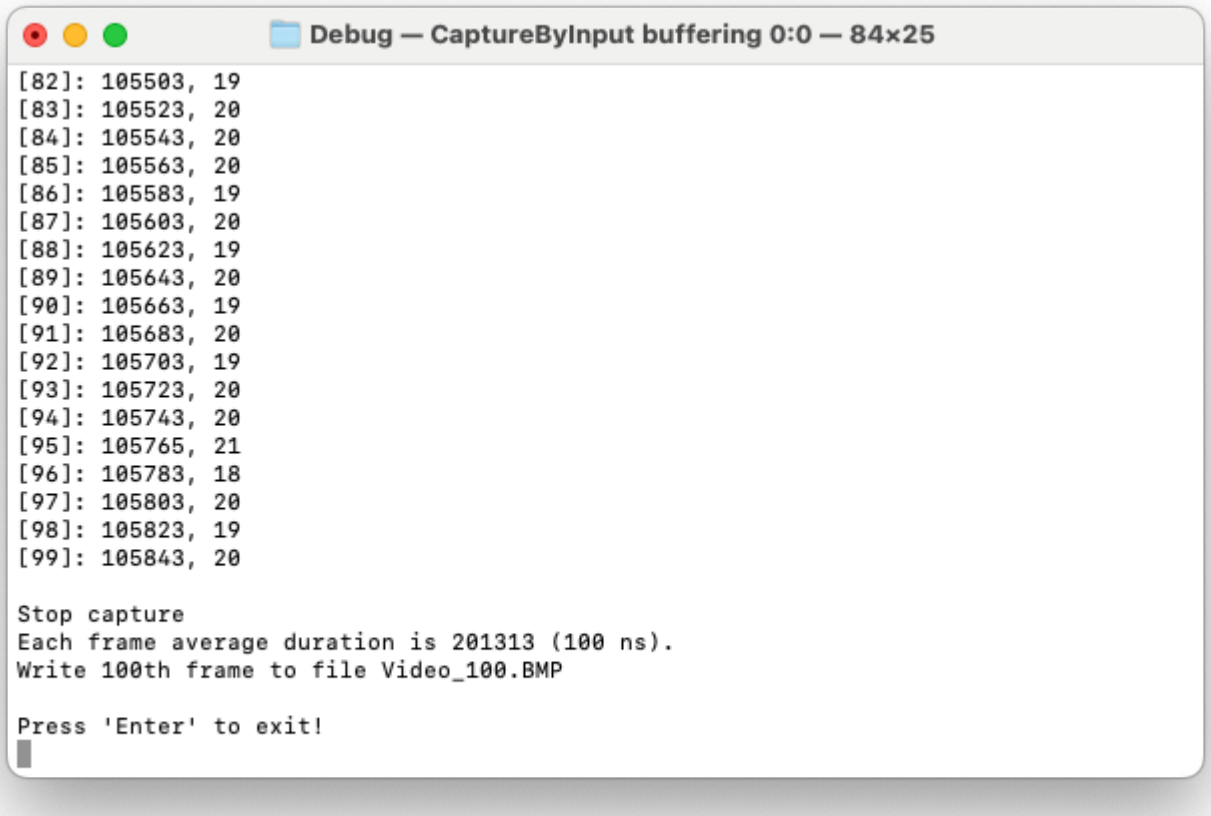
1. Capturing video data according to the format of input signals
2. Saving the last frame of video data as a BMP image
3. Channel selection via command parameters when multiple devices are available

Call Logic:

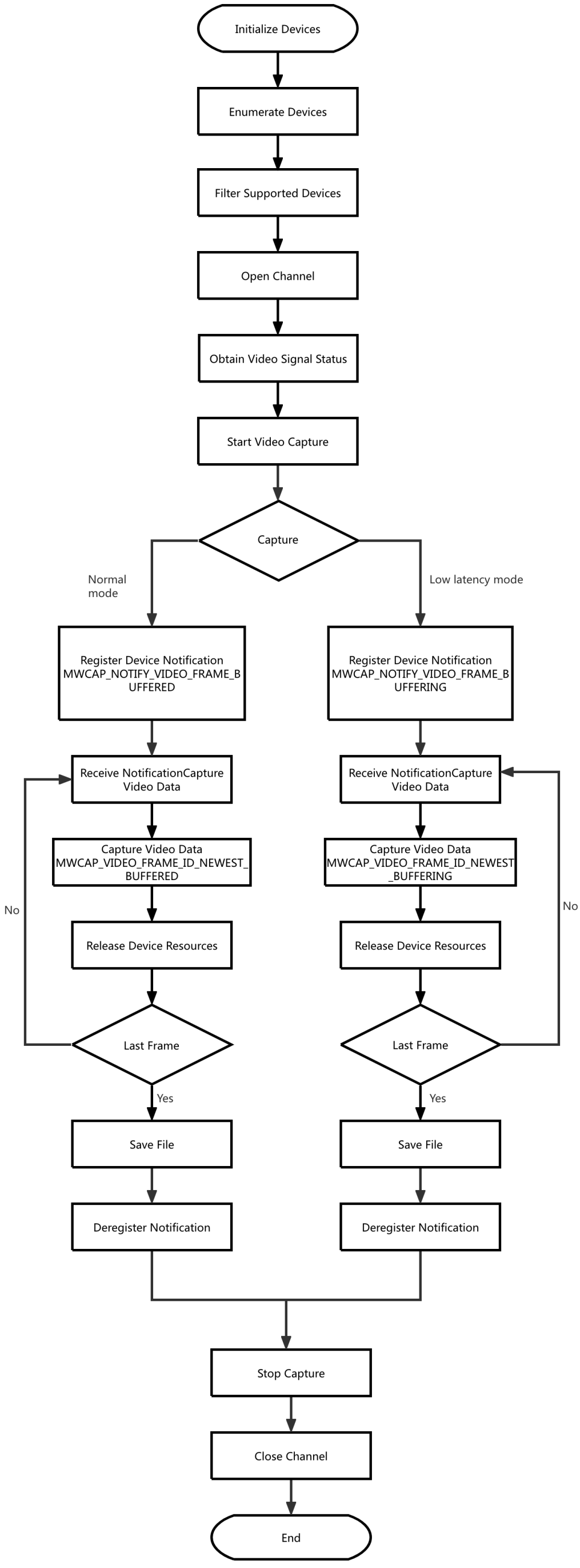
1. Obtain version information, initialize, and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Filter supported devices based on device and interface information: MWGetChannelInfoByIndex
3. Open the device:
    i. Open the device specified by command parameters: MWOpenChannel
    ii. When no device is specified in command parameters, open the first available channel by default: MWGetDevicePath, MWOpenChannelByPath
4. Retrieve video signal status: MWGetVideoSignalStatus
5. Start video capture: MWStartVideoCapture
6. Create event: MWCreateEvent
7. Register for message notifications: MWRegisterNotify MWCAP_NOTIFY_VIDEO_FRAME_BUFFERED (normal mode), MWCAP_NOTIFY_VIDEO_FRAME_BUFFERING (low-latency mode)
8. Wait for event: MWTryWaitEvent, MWWaitEvent
9. Capture video data: MWCaptureVideoFrameToVirtualAddressEx MWCAP_VIDEO_FRAME_ID_NEWEST_BUFFERED (normal mode), MWCAP_VIDEO_FRAME_ID_NEWEST_BUFFERING (low-latency mode)
10. Retrieve current capture status: MWGetVideoCaptureStatus
11. Repeat steps 8, 9, and 10 until the last frame
12. Save file
13. Deregister message notifications: MWUnregisterNotify
14. Close event: MWCloseEvent
15. Stop capture: MWStopVideoCapture
16. Release resources: MWCloseChannel, MWCaptureExitInstance

Result:




Debug — CaptureByInput buffering 0:0 — 84×25
[82]: 105503, 19
[83]: 105523, 20
[84]: 105543, 20
[85]: 105563, 20
[86]: 105583, 19
[87]: 105603, 20
[88]: 105623, 19
[89]: 105643, 20
[90]: 105663, 19
[91]: 105683, 20
[92]: 105703, 19
[93]: 105723, 20
[94]: 105743, 20
[95]: 105765, 21
[96]: 105783, 18
[97]: 105803, 20
[98]: 105823, 19
[99]: 105843, 20
Stop capture
Each frame average duration is 201313 (100 ns).
Write 100th frame to file Video_100.BMP
Press 'Enter' to exit!

Flowchart:


Initialize Devices
Enumerate Devices
Filter Supported Devices
Open Channel
Obtain Video Signal Status
Start Video Capture
Capture
Normal mode
Low latency mode
Register Device Notification MWCAP_NOTIFY_VIDEO_FRAME_BUFFERED
Register Device Notification MWCAP_NOTIFY_VIDEO_FRAME_BUFFERING
Receive NotificationCapture Video Data
Receive NotificationCapture Video Data
Capture Video Data MWCAP_VIDEO_FRAME_ID_NEWEST_BUFFERED
Capture Video Data MWCAP_VIDEO_FRAME_ID_NEWEST_BUFFERING
No
No
Release Device Resources
Release Device Resources
Last Frame
Last Frame
Yes
Yes
Save File
Save File
Deregister Notification
Deregister Notification
Stop Capture
Close Channel
End

CaptureByTimer

Supported Hardware Devices: Pro Capture cards

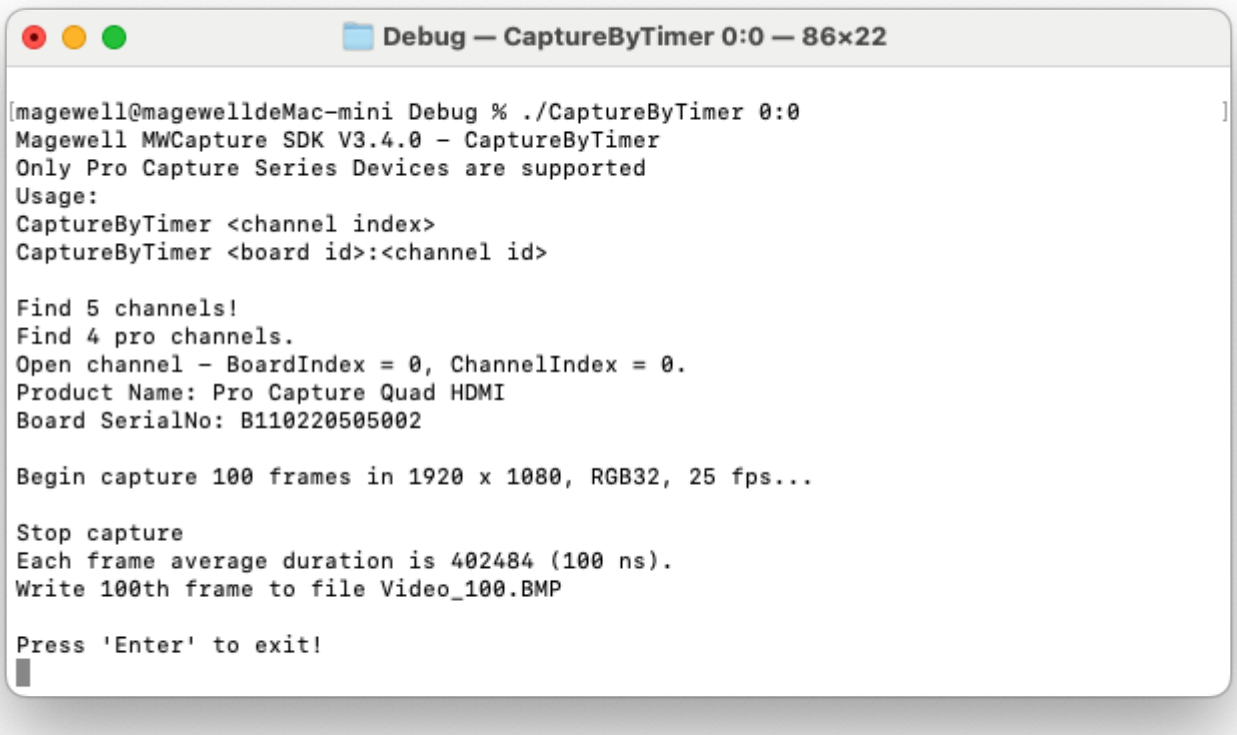
Contents Demonstrated in the Sample:

1. Capturing video data from input signals at a specified frame rate
2. Saving the last frame of video data as a BMP image
3. Channel selection via command parameters when multiple devices are available

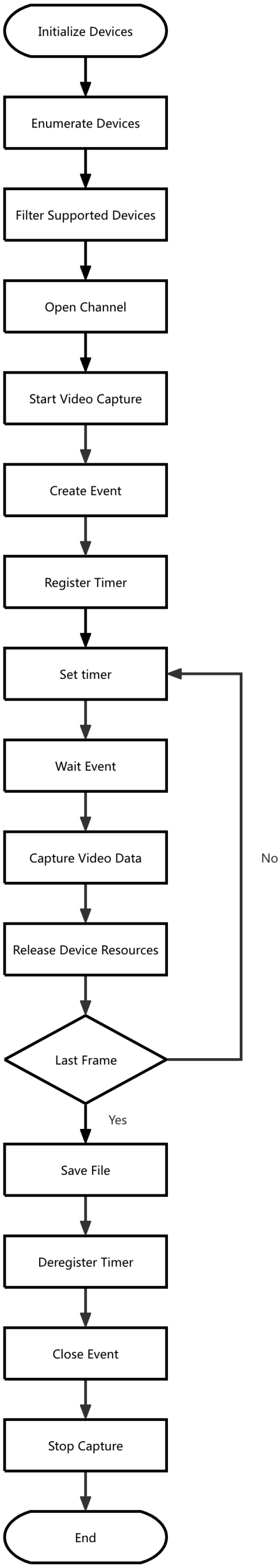
Call Logic:

1. Obtain version information, initialize, and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Filter supported devices based on device and interface information: MWGetChannelInfoByIndex
3. Open the device:
    i. Open the device specified by command parameters: MWOpenChannel
    ii. When no device is specified in command parameters, open the first available channel by default: MWGetDevicePath, MWOpenChannelByPath
4. Start video capture: MWStartVideoCapture
5. Create event: MWCreateEvent
6. Register timer: MWRegisterTimer
7. Set time schedule: MWScheduleTimer
8. Wait for event: MWTryWaitEvent, MWWaitEvent
9. Capture video data: MWCaptureVideoFrameToVirtualAddressEx
10. Retrieve current capture status: MWGetVideoCaptureStatus
11. Repeat steps 7, 8, 9, and 10 until the last frame
12. Save file
13. Deregister timer: MWUnregisterTimer
14. Close event: MWCloseEvent
15. Stop capture: MWStopVideoCapture
16. Release resources: MWCloseChannel, MWCaptureExitInstance

Result:



Flowchart:


Initialize Devices
Enumerate Devices
Filter Supported Devices
Open Channel
Start Video Capture
Create Event
Register Timer
Set timer
Wait Event
Capture Video Data
Release Device Resources
Last Frame
No
Yes
Save File
Deregister Timer
Close Event
Stop Capture
End

GetANCPacket

Supported Hardware Devices: Pro Capture cards supporting SDI signals

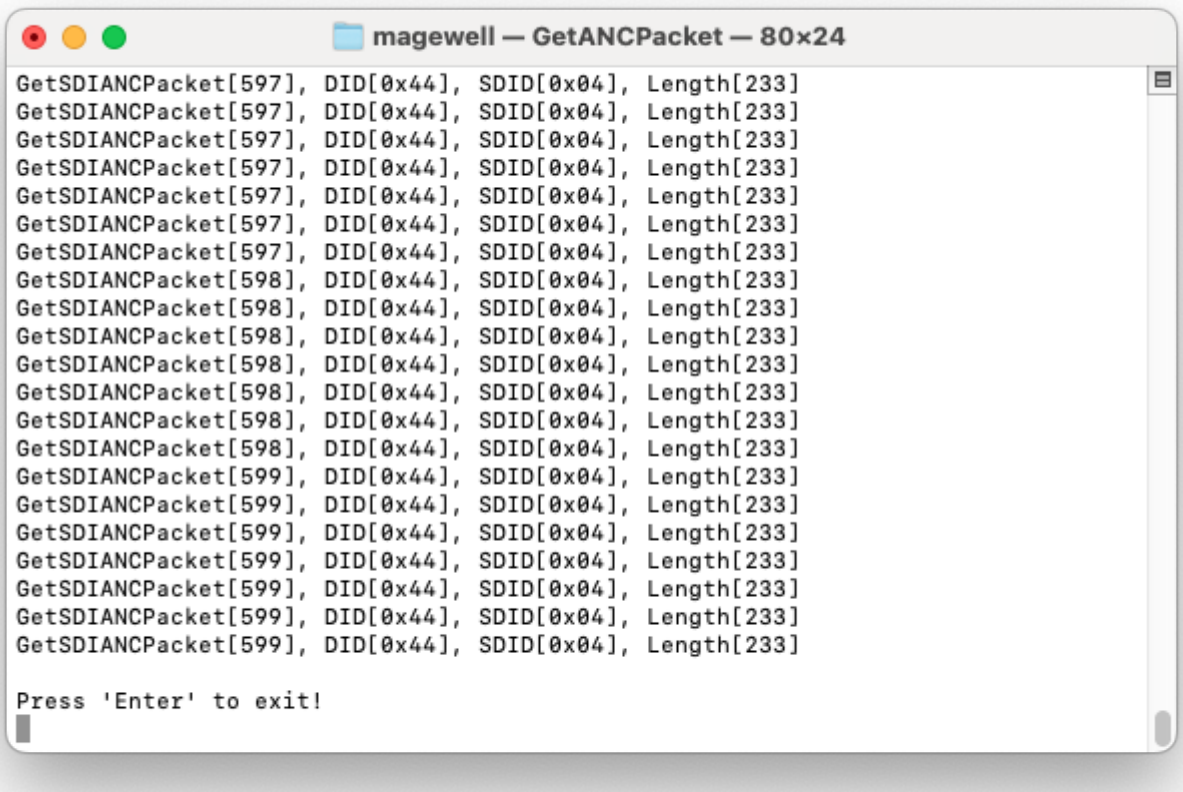
Contents Demonstrated in the Sample:

1. Retrieving ANC data packets from SDI signals
2. Channel selection via command parameters when multiple devices are available

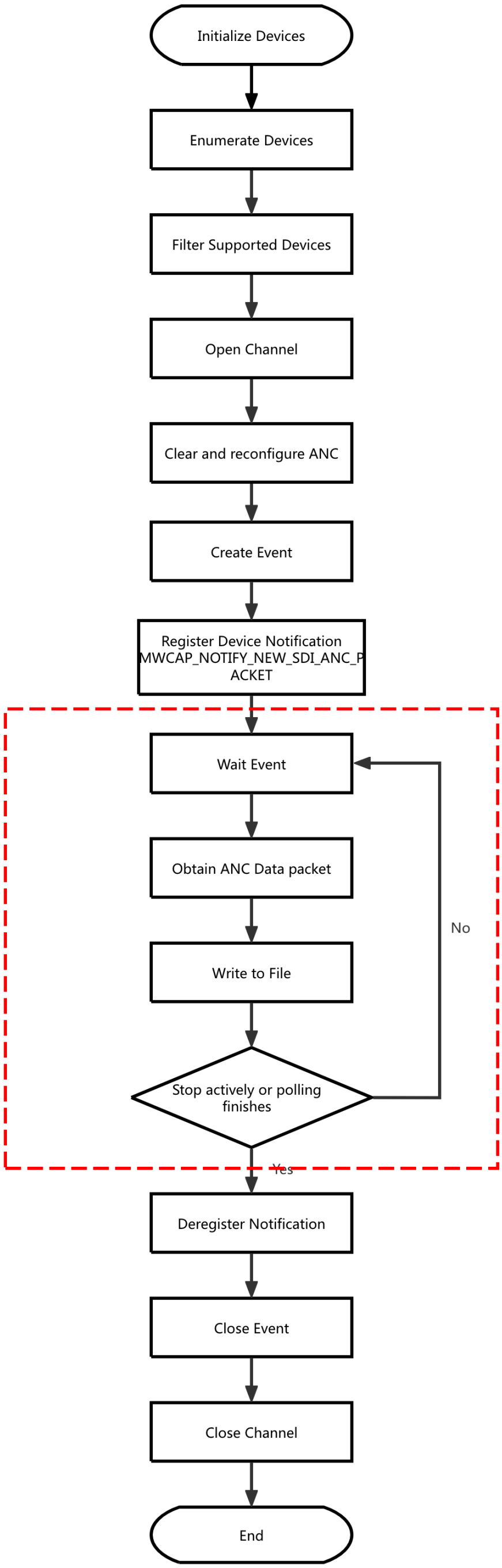
Call Logic:

1. Obtain version information, initialize, and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Filter supported devices based on device and interface information: MWGetChannelInfoByIndex
3. Open the device:
    i. Open the device specified by command parameters: MWOpenChannel
    ii. When no device is specified in command parameters, open the first available channel by default: MWGetDevicePath，MWOpenChannelByPath
4. Clear and reconfigure ANC: MWCaptureSetSDIANCType
5. Create event: MWCreateEvent
6. Register for message notifications: MWRegisterNotify
7. Wait for event: MWTryWaitEvent, MWWaitEvent
8. Retrieve ANC data packet: MWCaptureGetSDIANCPacket
9. Write data to binary file
10. Repeat steps 7, 8, and 9 until polling ends or user actively stops
11. Deregister message notifications: MWUnregisterNotify
12. Close event: MWCloseEvent
13. Release resources: MWCloseChannel, MWCaptureExitInstance

Result:



Flowchart:


Initialize Devices
Enumerate Devices
Filter Supported Devices
Open Channel
Clear and reconfigure ANC
Create Event
Register Device Notification
MWCAP_NOTIFY_NEW_SDI_ANC_P
ACKET
Wait Event
Obtain ANC Data packet
Write to File
Stop actively or polling
finishes
No
Yes
Deregister Notification
Close Event
Close Channel
End

HDMIInfoFrame

Supported Hardware Devices: Pro Capture cards supporting HDMI signals, USB Capture devices supporting HDMI signals

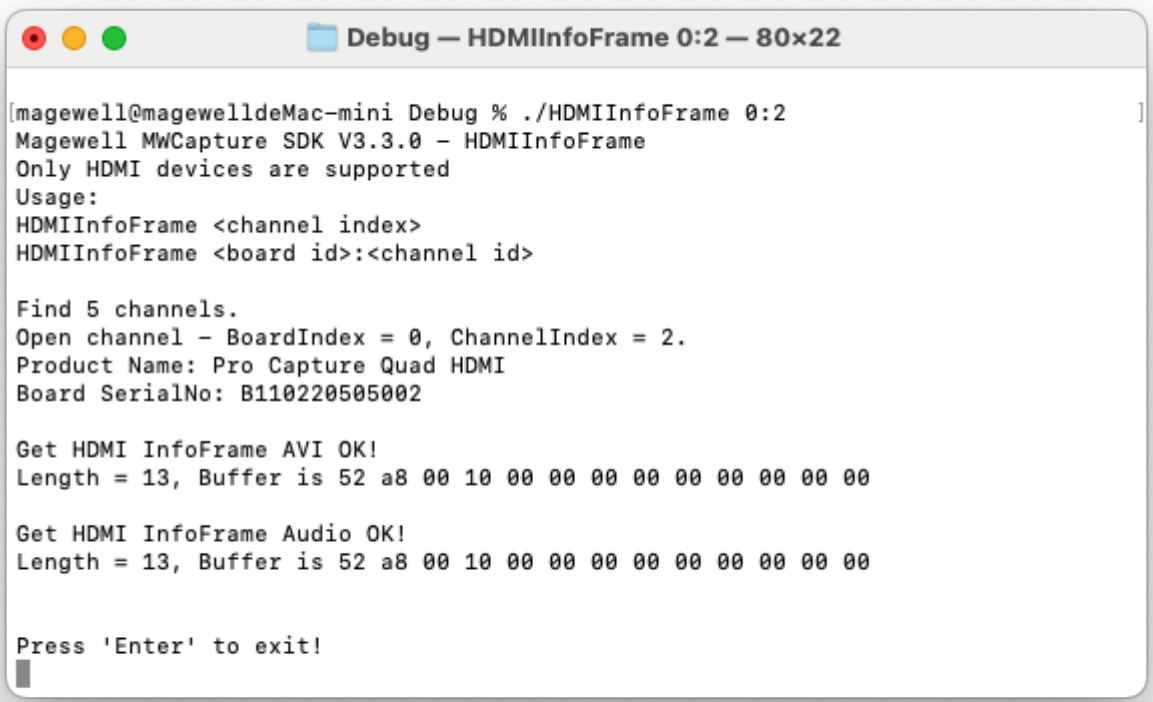
Contents Demonstrated in the Sample:

1. Detecting and retrieving HDMI signal status and HDMI frame information
2. Channel selection via command parameters when multiple devices are available

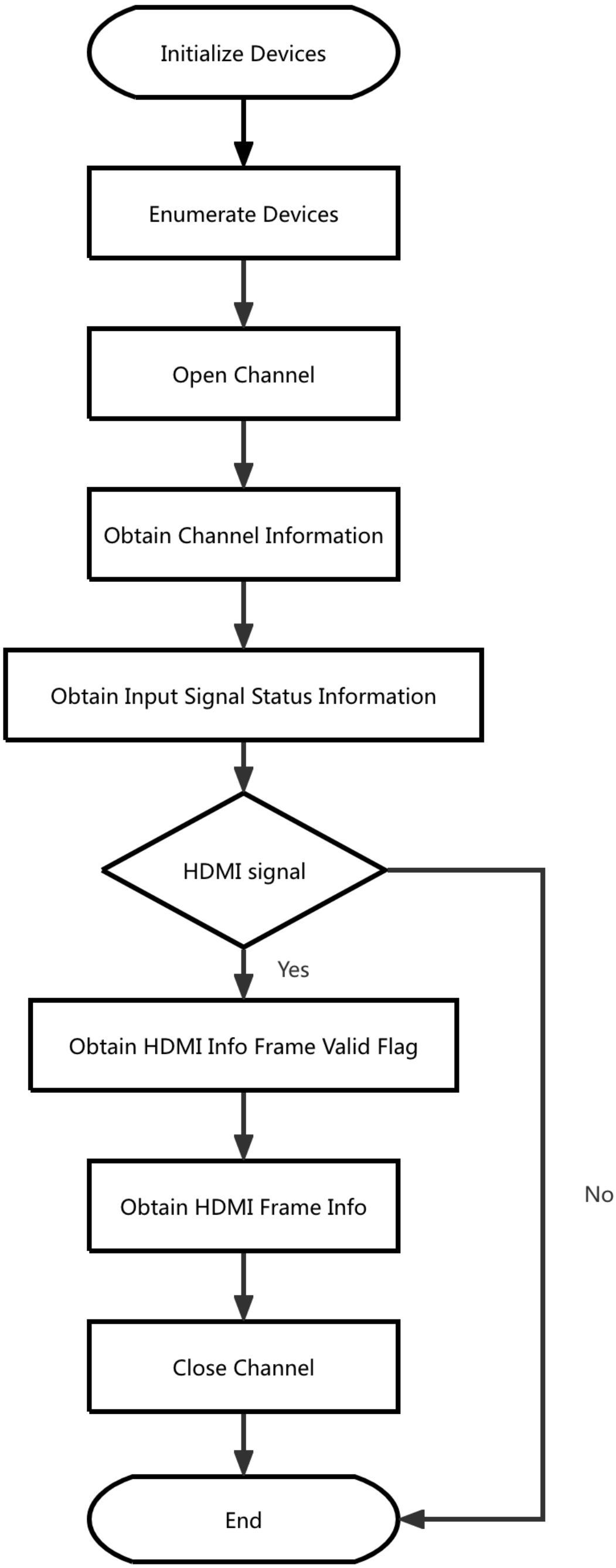
Call Logic:

1. Obtain version information, initialize, and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Open the device:
    i. Open the device specified by command parameters: MWOpenChannel
    ii. When no device is specified in command parameters, open the first available channel by default: MWGetDevicePath, MWOpenChannelByPath
3. Retrieve channel information: MWGetChannelInfo
4. Retrieve input signal status information: MWGetInputSpecificStatus
5. Check if HDMI information frame exists: MWGetHDMIInfoFrameValidFlag
6. Retrieve and output HDMI frame information: MWGetHDMIInfoFramePacket
7. Release resources: MWCloseChannel, MWCaptureExitInstance

Result:



Flowchart:


Initialize Devices
Enumerate Devices
Open Channel
Obtain Channel Information
Obtain Input Signal Status Information
HDMI signal
Yes
No
Obtain HDMI Info Frame Valid Flag
Obtain HDMI Frame Info
Close Channel
End

InputSignal

Supported Hardware Devices: Pro Capture cards, USB Capture devices

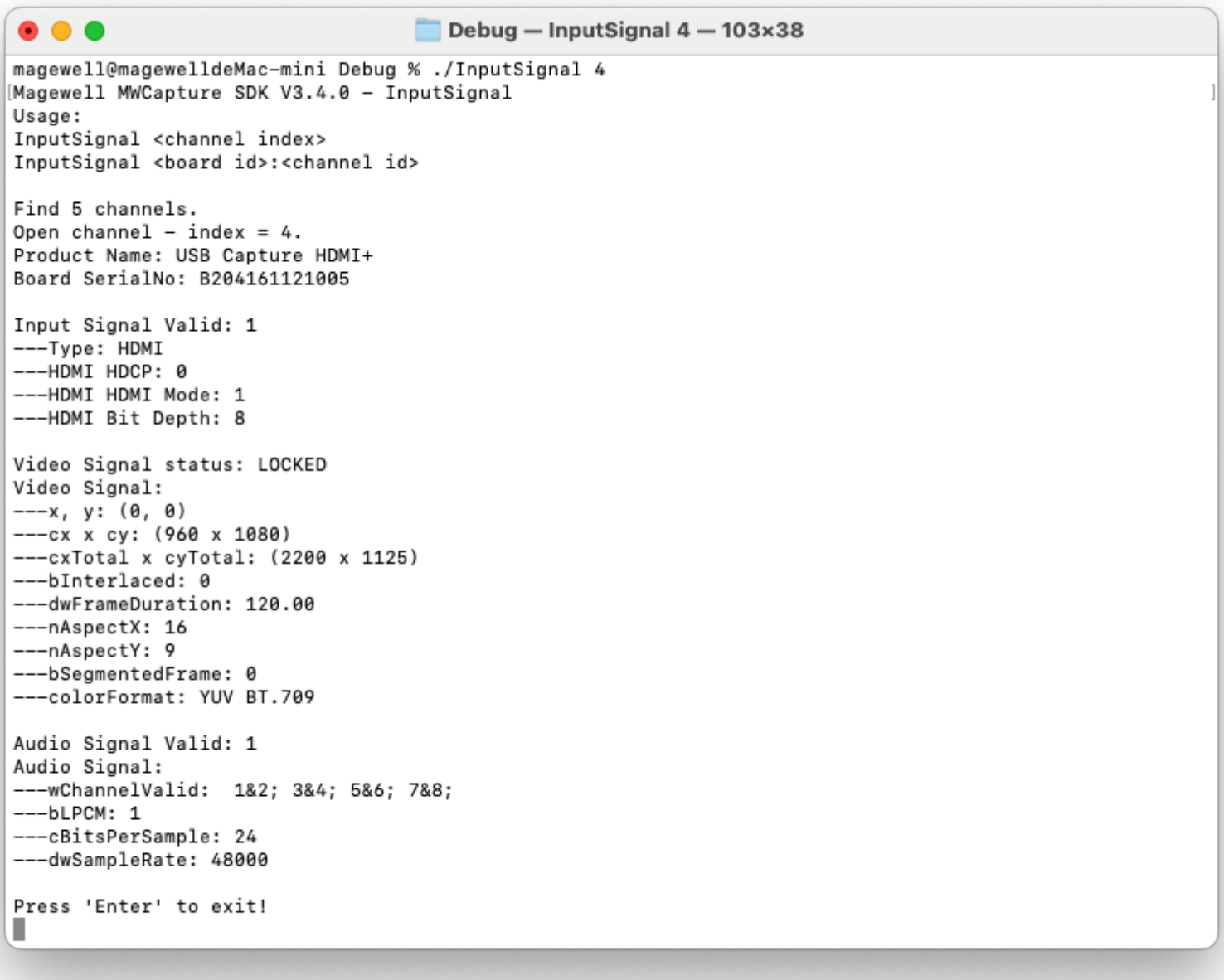
Contents Demonstrated in the Sample:

1. Retrieving detailed format information of input audio and video signals
2. Channel selection via command parameters when multiple devices are available

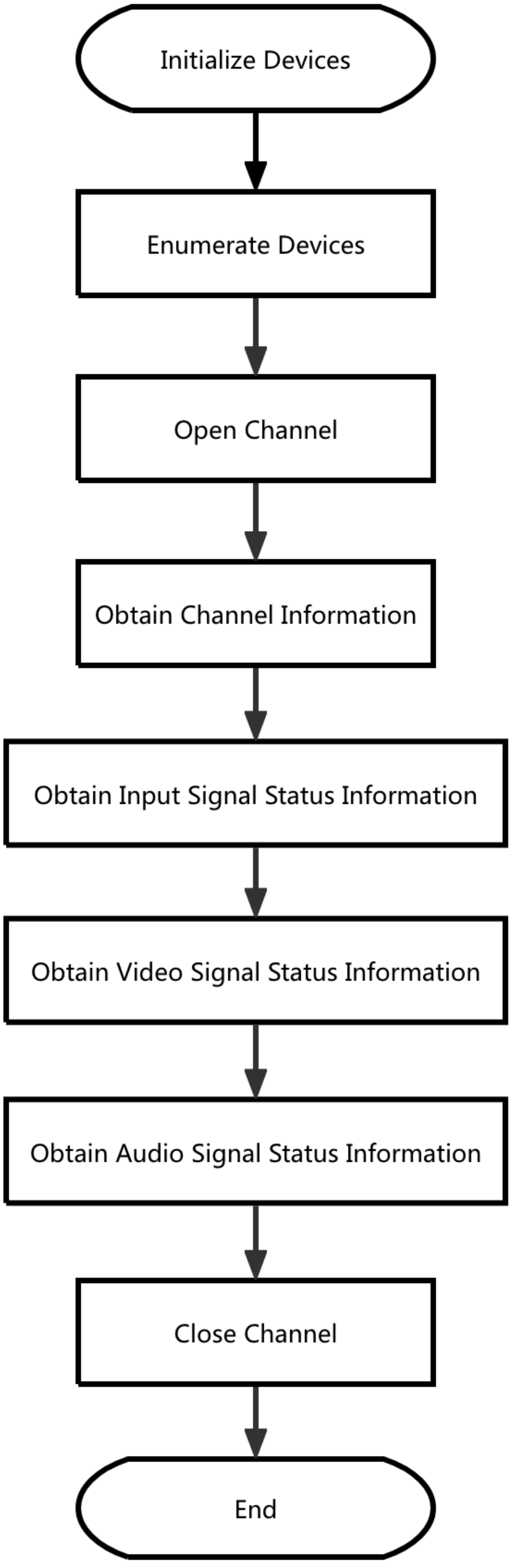
Call Logic:

1. Obtain version information, initialize, and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Open the device:
    i. Open the device specified by command parameters: MWOpenChannel
    ii. When no device is specified in command parameters, open the first available channel by default: MWGetDevicePath, MWOpenChannelByPath
3. Retrieve channel information: MWGetChannelInfo
4. Retrieve input signal status information: MWGetInputSpecificStatus
5. Retrieve video signal status information: MWGetVideoSignalStatus
6. Retrieve audio signal status information: MWGetAudioSignalStatus
7. Release resources: MWCloseChannel, MWCaptureExitInstance

Result:



Flowchart:


Initialize Devices
Enumerate Devices
Open Channel
Obtain Channel Information
Obtain Input Signal Status Information
Obtain Video Signal Status Information
Obtain Audio Signal Status Information
Close Channel
End

InputSource

Supported Hardware Devices: Pro Capture cards, USB Capture devices

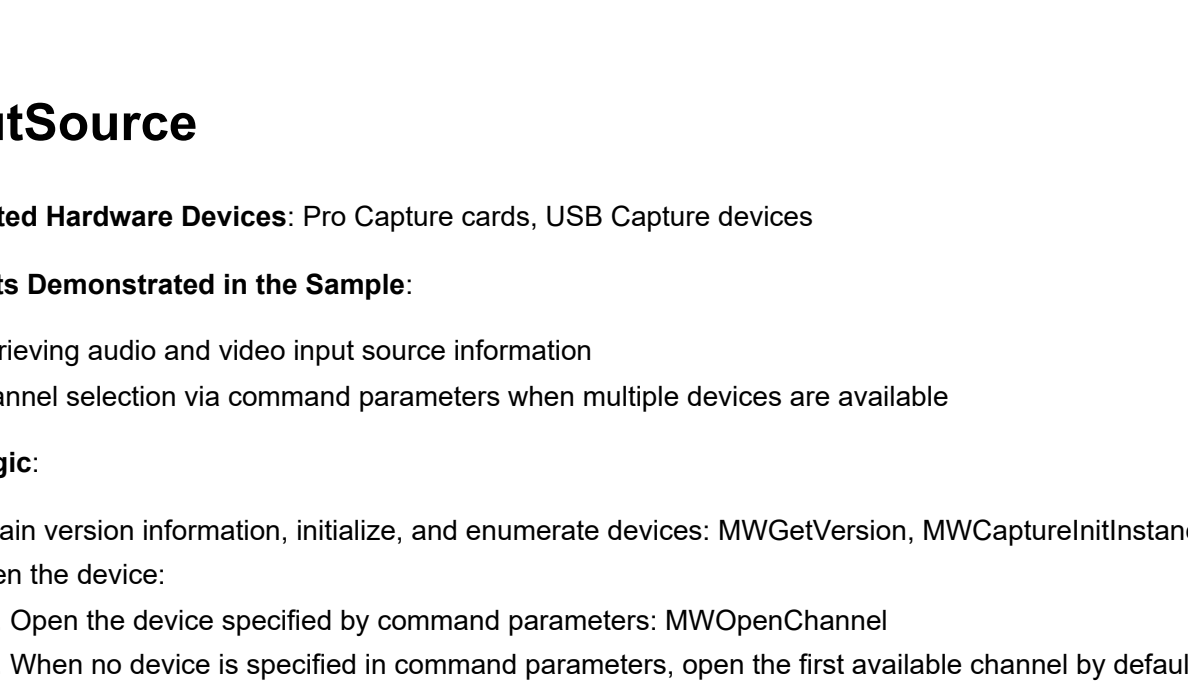
Contents Demonstrated in the Sample:

1. Retrieving audio and video input source information
2. Channel selection via command parameters when multiple devices are available

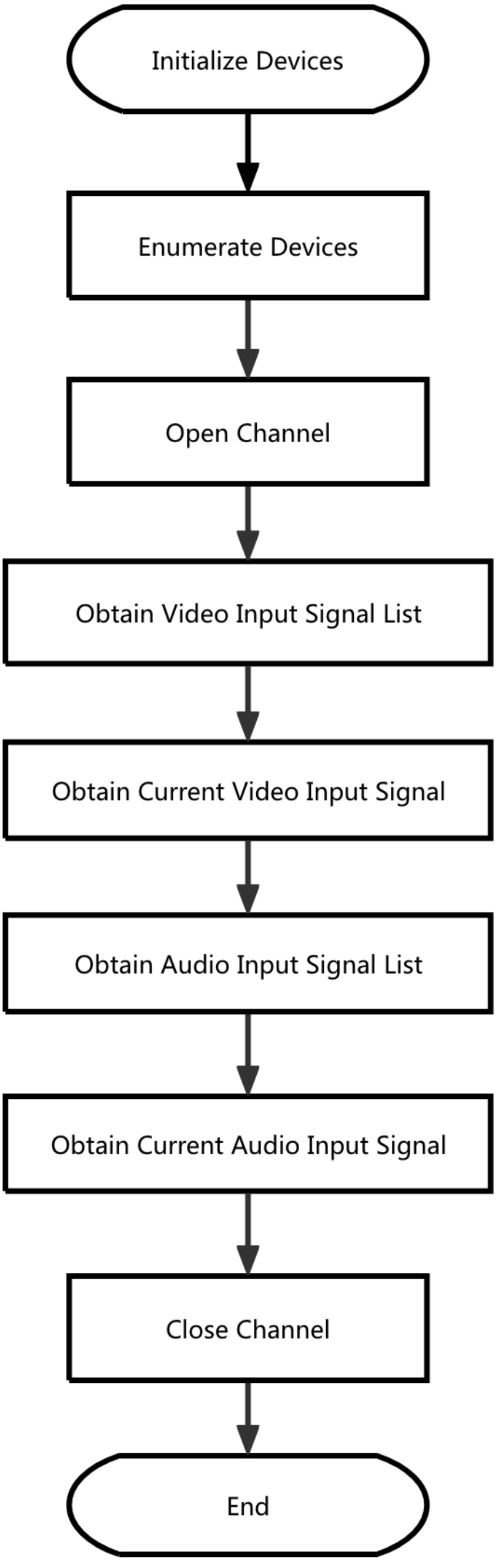
Call Logic:

1. Obtain version information, initialize, and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Open the device:
    i. Open the device specified by command parameters: MWOpenChannel
    ii. When no device is specified in command parameters, open the first available channel by default: MWGetDevicePath, MWOpenChannelByPath
3. Retrieve channel information: MWGetChannelInfo
4. Retrieve video input array and video input source: MWGetVideoInputSourceArray, MWGetVideoInputSource
5. Retrieve audio input array and audio input source: MWGetAudioInputSourceArray, MWGetAudioInputSource
6. Release resources: MWCloseChannel, MWCaptureExitInstance

Result:



Flowchart:


Initialize Devices
Enumerate Devices
Open Channel
Obtain Video Input Signal List
Obtain Current Video Input Signal
Obtain Audio Input Signal List
Obtain Current Audio Input Signal
Close Channel
End

InputSignalNotify

Supported Hardware Devices: Pro Capture cards

Contents Demonstrated in the Sample:

1. Monitoring capture card input signal changes
2. Channel selection via command parameters when multiple devices are available

Call Logic:

1. Obtain version information, initialize, and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Open the device:
    i. Open the device specified by command parameters: MWOpenChannel
    ii. When no device is specified in command parameters, open the first available channel by default: MWGetDevicePath, MWOpenChannelByPath
3. Retrieve channel information: MWGetChannelInfo
4. Create video input signal change monitoring thread
5. Create audio input signal change monitoring thread
6. Exit video input signal change monitoring thread
7. Exit audio input signal change monitoring thread
8. Release resources: MWCloseChannel, MWCaptureExitInstance

Video Input Signal Change Monitoring Thread:

1. Create event: MWCreateEvent
2. Register notification: MWRegisterNotify
3. Wait for event: MWMultiWaitEvent
4. Retrieve video signal status: MWGetVideoSignalStatus
5. Unregister notification: MWUnregisterNotify
6. Close event: MWCloseEvent

Audio Input Signal Change Monitoring Thread:

1. Create event: MWCreateEvent
2. Register notification: MWRegisterNotify
3. Wait for event: MWMultiWaitEvent
4. Retrieve audio signal status: MWGetAudioSignalStatus
5. Unregister notification: MWUnregisterNotify
6. Close event: MWCloseEvent

Result:


Debug — -zsh — 107×64
magewell@magewelldeMac-mini Debug % ./InputSignalNotify 0:0
Magewell MWCapture SDK V3.4.0 - InputSignalNotify
Usage:
InputSignalNotify <channel index>
InputSignalNotify <board id>:<channel id>
Find 5 channels.
Open channel - BoardIndex = 0, ChannelIndex = 0.
Product Name: Pro Capture Quad HDMI
Board SerialNo: B110220505002
Press 'Enter' to exit!
--- Video signal changed ---
Video Signal status: UNSUPPORTED
--- Video signal changed end ---
--- Video signal changed ---
--- Audio signal changed ---
Video Signal status: LOCKED
Video Signal:
---x, y: (260, 58)
---cx x cy: (1280 x 1440)
---cxTotal x cyTotal: (1980 x 1500)
---bInterlaced: 1
---dwFrameDuration: 50.00
---nAspectX: 16
---nAspectY: 9
---bSegmentedFrame: 0
---colorFormat: YUV BT.709
--- Video signal changed end ---
Audio Signal Valid: 1
Audio Signal:
---wChannelValid: 1&2;
---bLPCM: 1
---cBitsPerSample: 24
---dwSampleRate: 44100
--- Audio signal changed end ---
--- Video signal changed ---
Video Signal status: LOCKED
Video Signal:
---x, y: (260, 25)
---cx x cy: (1280 x 720)
---cxTotal x cyTotal: (1980 x 750)
---bInterlaced: 0
---dwFrameDuration: 50.00
---nAspectX: 16
---nAspectY: 9
---bSegmentedFrame: 0
---colorFormat: YUV BT.709
--- Video signal changed end ---
Vudio signal thread exist
Audio signal thread exist
magewell@magewelldeMac-mini Debug %

Flowchart:

Initialize Devices
Enumerate Devices
Filter Supported Devices
Open Channel
Monitor Video Signal Change
Monitor Audio Signal Change
Close Channel
End
Video Signal Change Monitoring Thread
Create Event
Register Device Notification MWCAP_NOTIFY_VIDEO_SIGNAL_CHANGE
Wait for Event
Obtain Video Signal Status
Deregister Notification
Close Event
Audio Signal Change Monitoring Thread
Create Event
Register Device Notification MWCAP_NOTIFY_AUDIO_SIGNAL_CHANGE
Wait for Event
Obtain Audio Signal Status
Deregister Notification
Close Event

ReadWriteEDID

Supported Hardware Devices: Pro/USB Capture devices supporting HDMI signals

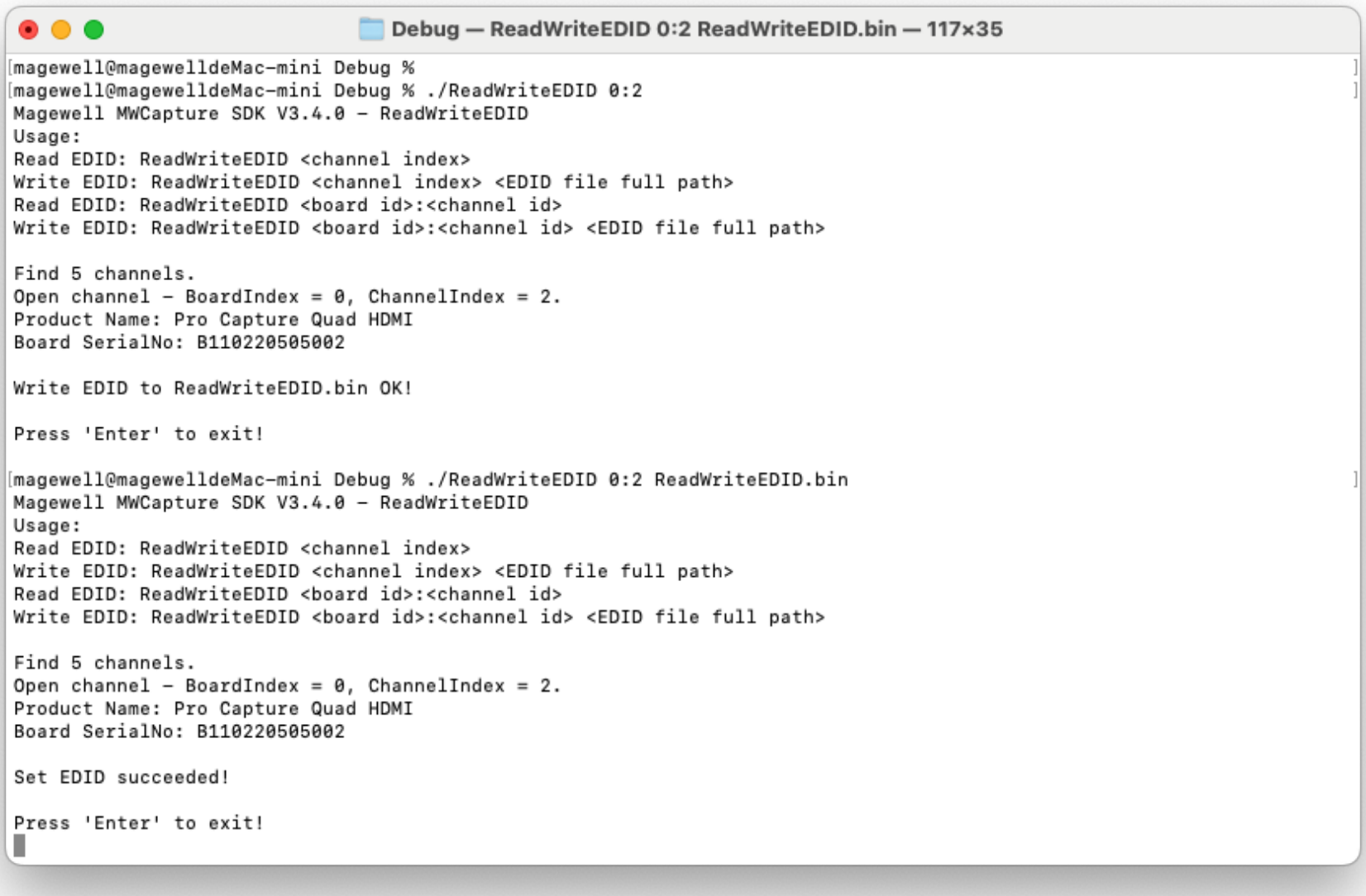
Contents Demonstrated in the Sample:

1. Export EDID from a specified channel to a binary file
2. Import a binary file (EDID) from a specified path and write it to a specified channel

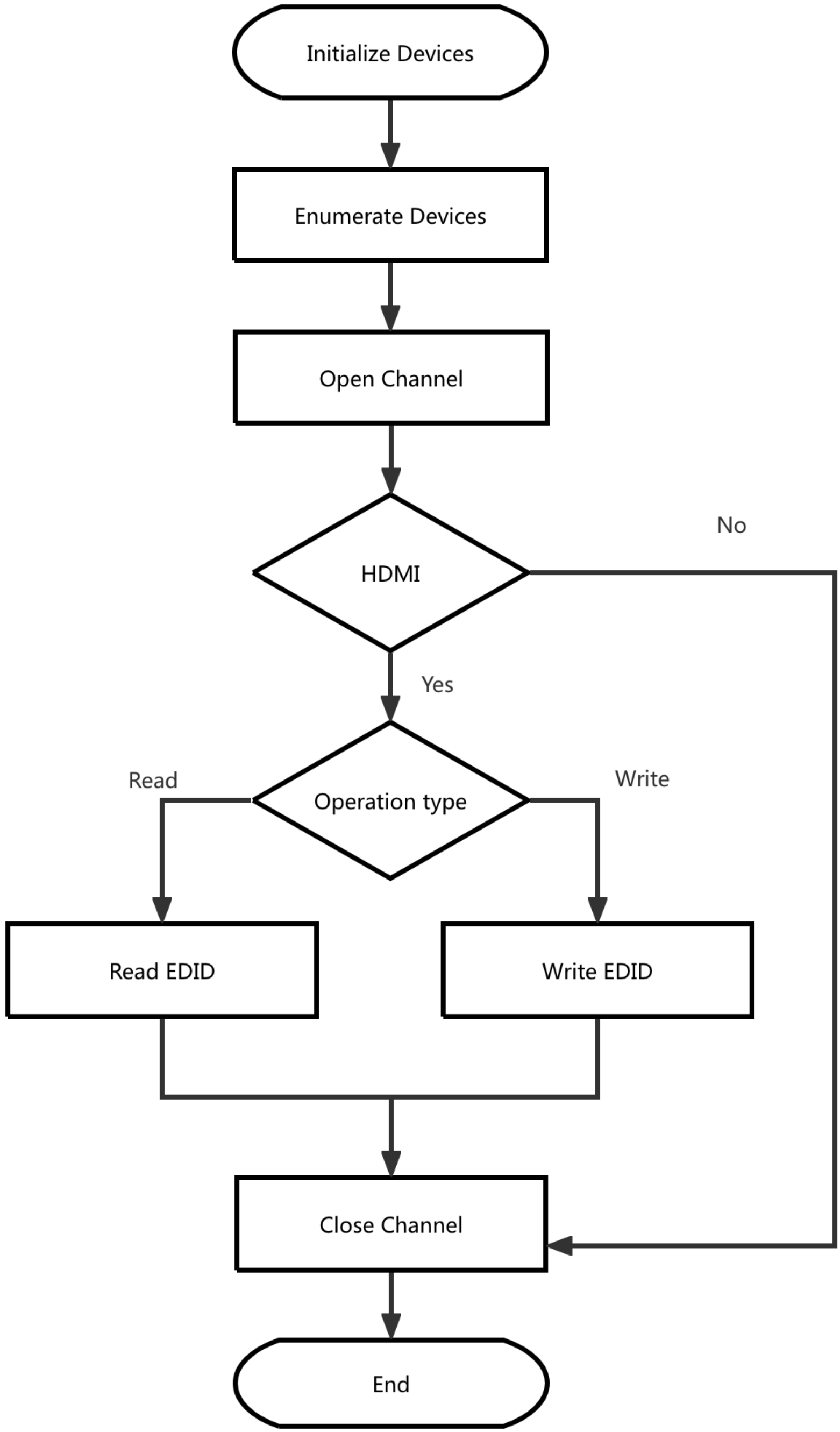
Call Logic:

1. Obtain version information, initialize, and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Open the device:
    i. Open the device specified by command parameters: MWOpenChannel
    ii. When no device is specified in command parameters, open the first available channel by default: MWGetDevicePath, MWOpenChannelByPath
3. Retrieve channel information and filter devices: MWGetChannelInfo, MWGetVideoInputSource, MWGetAudioInputSource
4. Determine based on command parameters whether to export the current channel information to a binary file or import binary file information and write it to the current channel:
    i. Save EDID information from the specified channel to a binary file:
        i. Retrieve EDID information: MWGetEDID
        ii. Write to binary file
    ii. Read binary file information and write to the specified channel:
        i. Read binary file: fread
        ii. Write EDID to current channel: MWSetEDID
5. Release resources: MWCloseChannel, MWCaptureExitInstance

Result:



Flowchart:


Initialize Devices
Enumerate Devices
Open Channel
HDMI
No
Yes
Read
Operation type
Write
Read EDID
Write EDID
Close Channel
End

USBDeviceDetect

Supported Hardware Devices: USB Capture devices

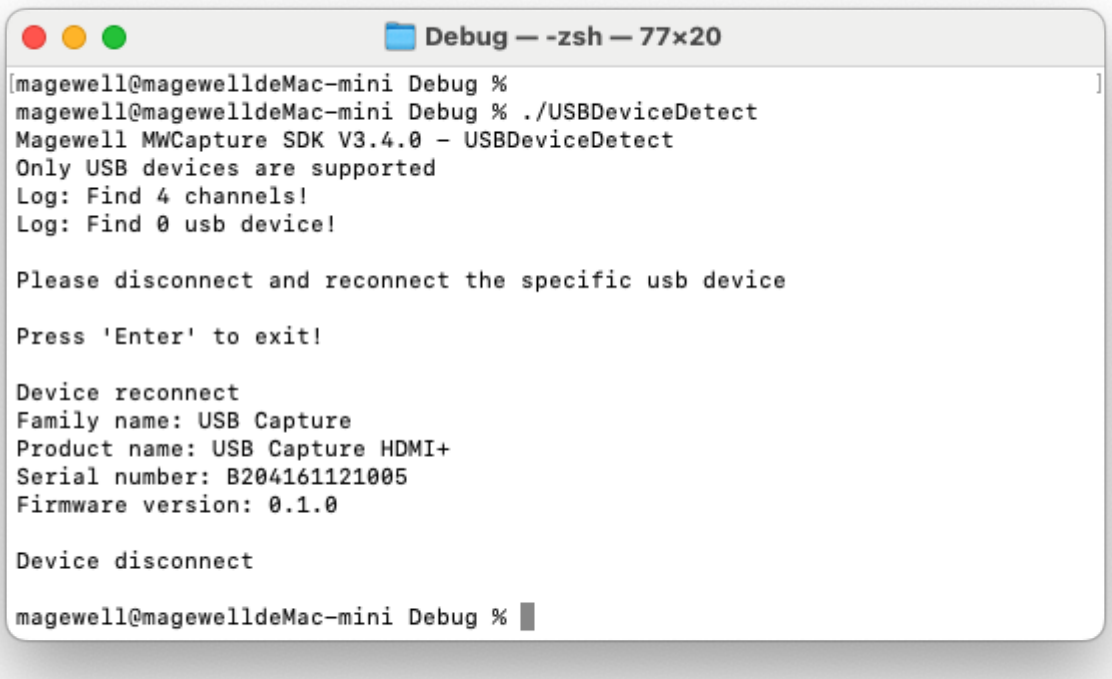
Contents Demonstrated in the Sample:

1. Monitoring hot-plugging events of USB capture devices

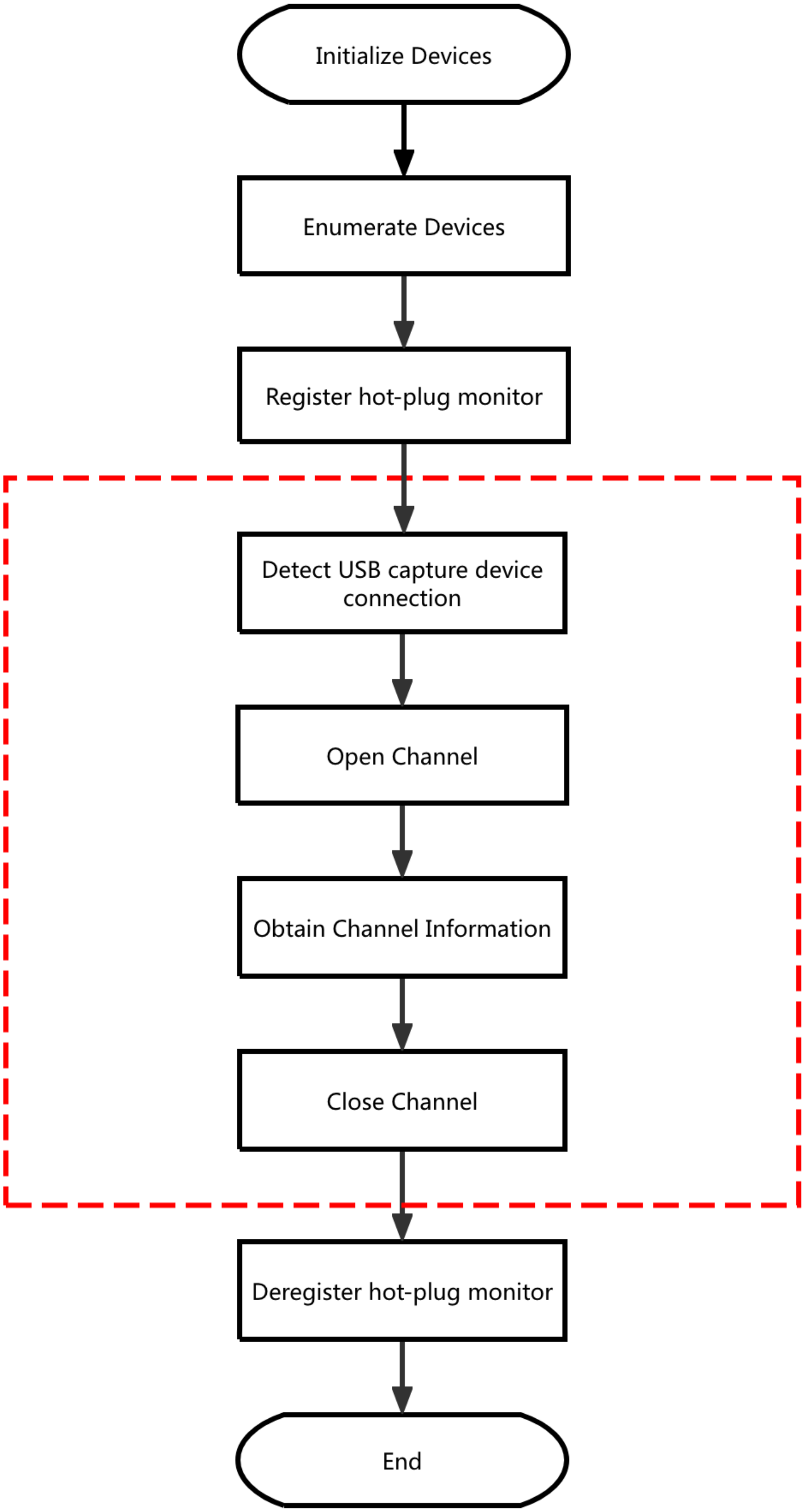
Call Logic:

1. Obtain version information, initialize, and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Declare hot-plug callback function: LPFN_HOT_PLUG_CALLBACK
3. Register hot-plug monitoring: MWUSBRegisterHotPlug
    i. When device connection is detected:
        i. Open channel: MWOpenChannelByPath
        ii. Retrieve channel information: MWGetChannelInfo
        iii. Close channel
    ii. When device disconnection is detected
4. Unregister hot-plug monitoring: MWUSBUnRegisterHotPlug
5. Release resources: MWCloseChannel, MWCaptureExitInstance

Result:



Flowchart:


Initialize Devices
Enumerate Devices
Register hot-plug monitor
Detect USB capture device connection
Open Channel
Obtain Channel Information
Close Channel
Deregister hot-plug monitor
End

SetUSBCaptureFourcc

Supported Hardware Devices: USB Capture devices

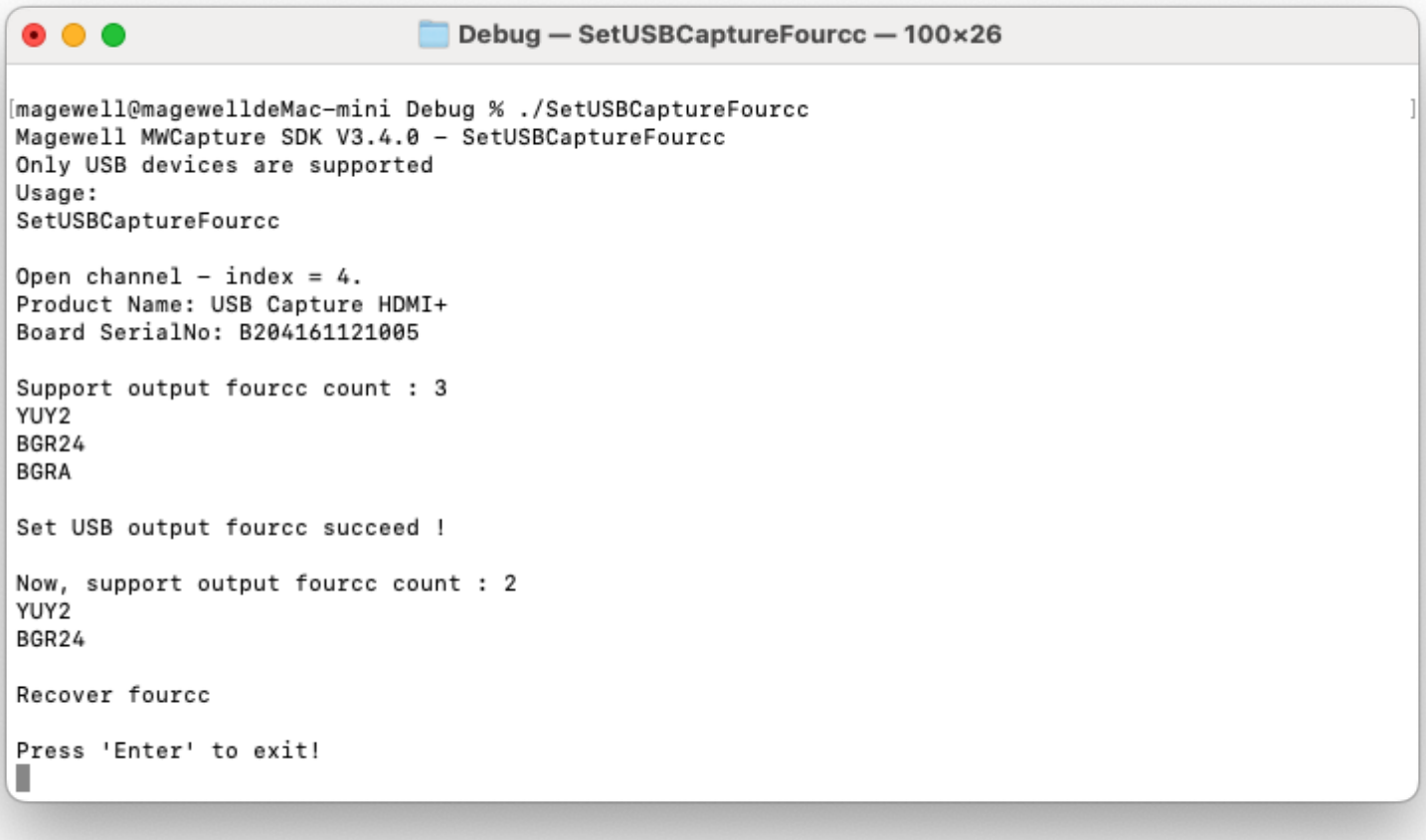
Contents Demonstrated in the Sample:

1. Reading and configuring the color space of USB capture devices

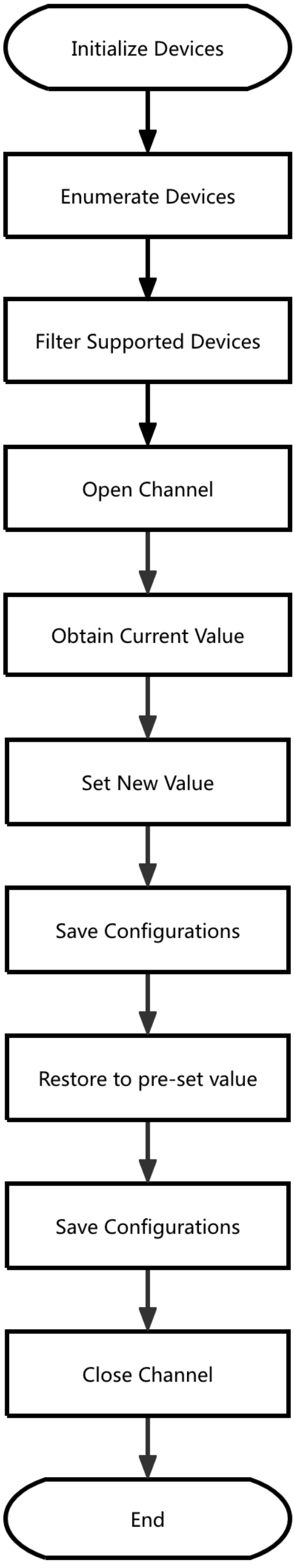
Call Logic:

1. Obtain version information, initialize, and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Retrieve device information and filter devices: MWGetChannelInfoByIndex
3. Open the device: MWOpenChannelByPath
4. Read the current value: MWUSBGetVideoOutputFOURCC
5. Set to new content: MWUSBSetVideoOutputFOURCC
6. Save the settings: MWUSBSaveOptions
7. Restore to the previous value: MWUSBSetVideoOutputFOURCC
8. Save the settings: MWUSBSaveOptions
9. Release resources: MWCloseChannel, MWCaptureExitInstance

Result:



Flowchart:


Initialize Devices
Enumerate Devices
Filter Supported Devices
Open Channel
Obtain Current Value
Set New Value
Save Configurations
Restore to pre-set value
Save Configurations
Close Channel
End

Mp4Repair

Supported Video Files: Corrupted files recorded using the LibMWMp4 library

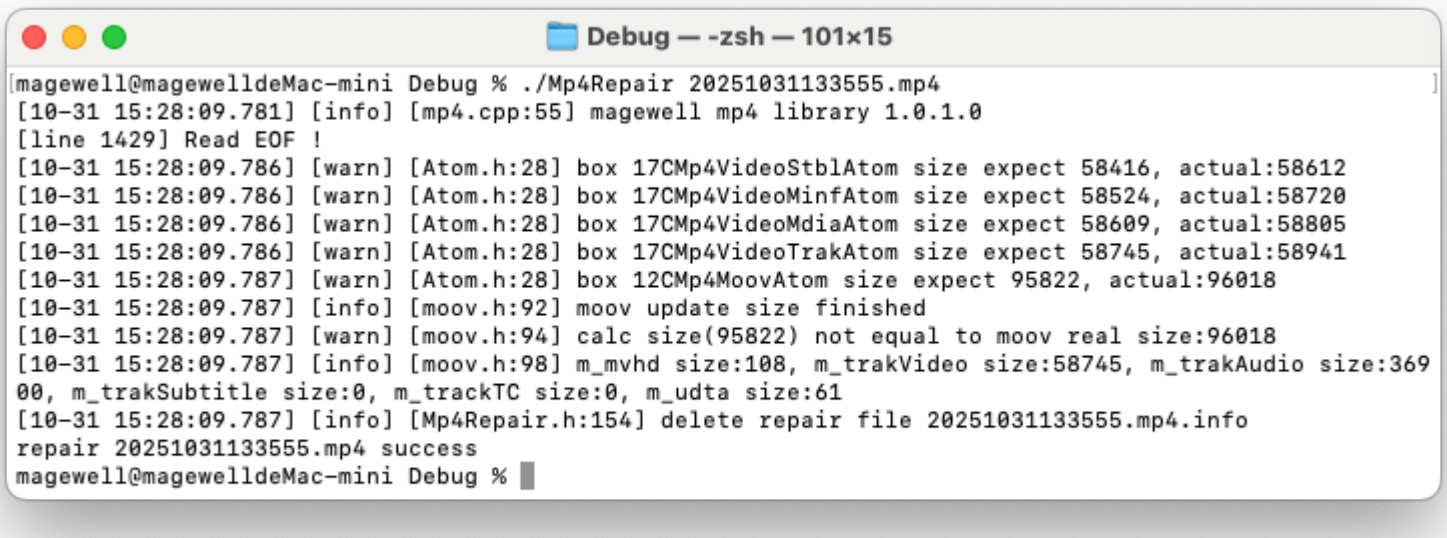
Contents Demonstrated in the Sample:

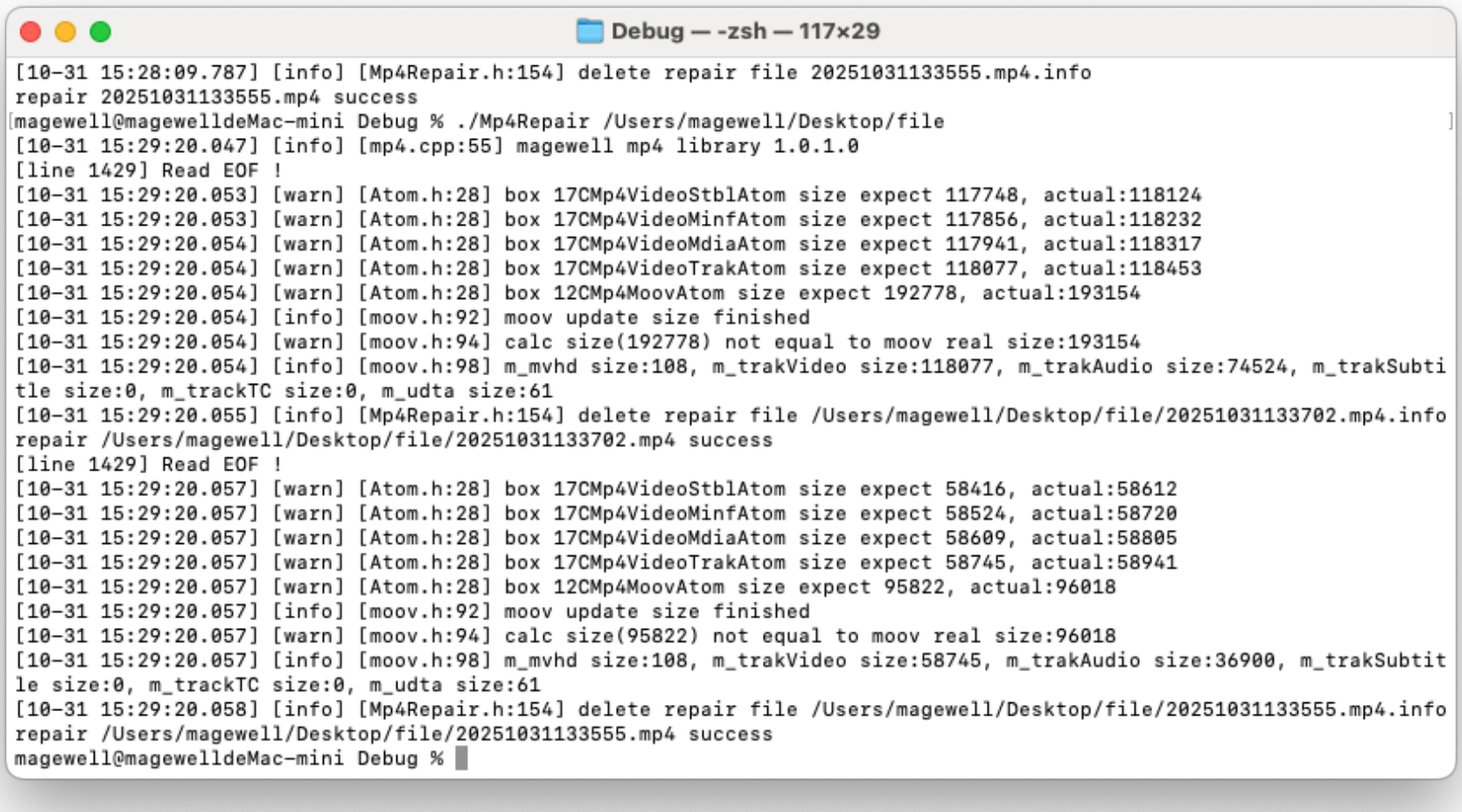
1. Repair a single corrupted file
2. Repair multiple corrupted files in a folder

Call Logic:

1. Repair a single corrupted file: Call the API to repair a single file
2. Repair multiple corrupted files in a folder:
    i. Enumerate files meeting the criteria in the folder
    ii. Call the API to repair each file

Result:





AVCapture

Supported Hardware Devices: Pro Capture cards

Contents Demonstrated in the Sample:

1. Preview captured video and monitor captured audio
2. Freely select video capture devices and their corresponding audio devices
3. Freely switch capture parameters: resolution, frame rate, color space
4. Support multi-channel capture: simultaneously capture one or two audio/video streams and render them separately.
5. Record captured audio and video data as FLV files.

Call Logic:

1. Obtain version information, initialize and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Filter supported devices based on device information and interface information: MWGetChannelInfoByIndex
3. Create a video rendering layer and overlay it on the current window: createPreviewLayer
4. Open channel: MWGetDevicePath, MWOpenChannelByPath
5. Create audio renderer: MWAudioRenderCreate, MWAudioRenderStart
6. Start video capture and rendering thread
7. Start audio capture and rendering thread
8. Enter main thread loop
9. Click menu to switch audio/video channels or capture parameters:
    i. Stop audio capture and rendering thread
    ii. Stop video capture and rendering thread
    iii. Destroy audio renderer: MWAudioRenderStop, MWAudioRenderDestroy
    iv. Close channel: MWCloseChannel
    v. Open channel and capture/render according to set channel and capture parameters: steps 4, 5, 6, 7
10. After clicking the close button, exit capture and release resources:
    i. Stop audio capture and rendering thread
    ii. Stop video capture and rendering thread
    iii. Destroy audio renderer: MWAudioRenderStop, MWAudioRenderDestroy
    iv. Close channel: MWCloseChannel
    v. Destroy instance: MWCaptureExitInstance.

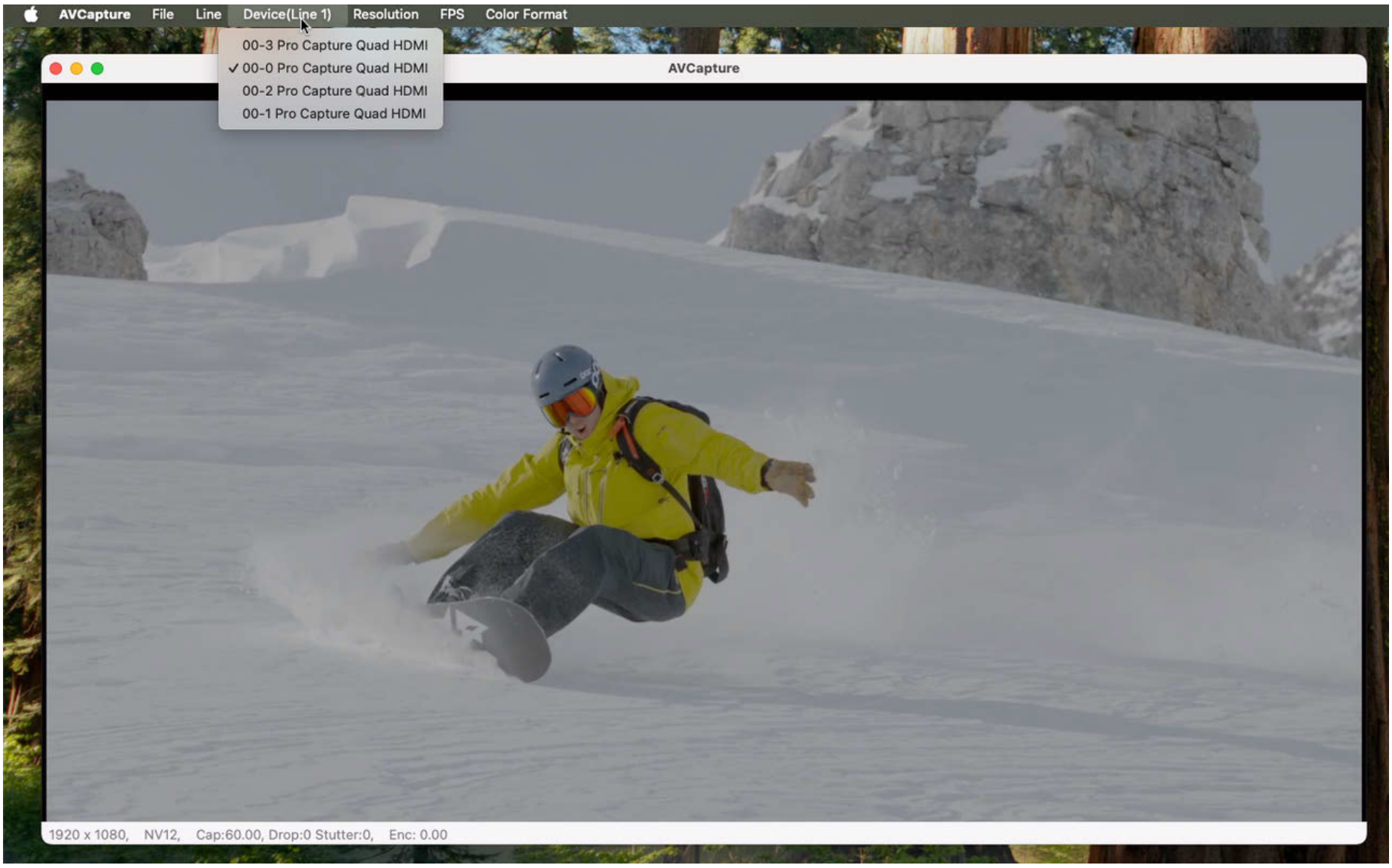
Video Capture and Rendering Thread:

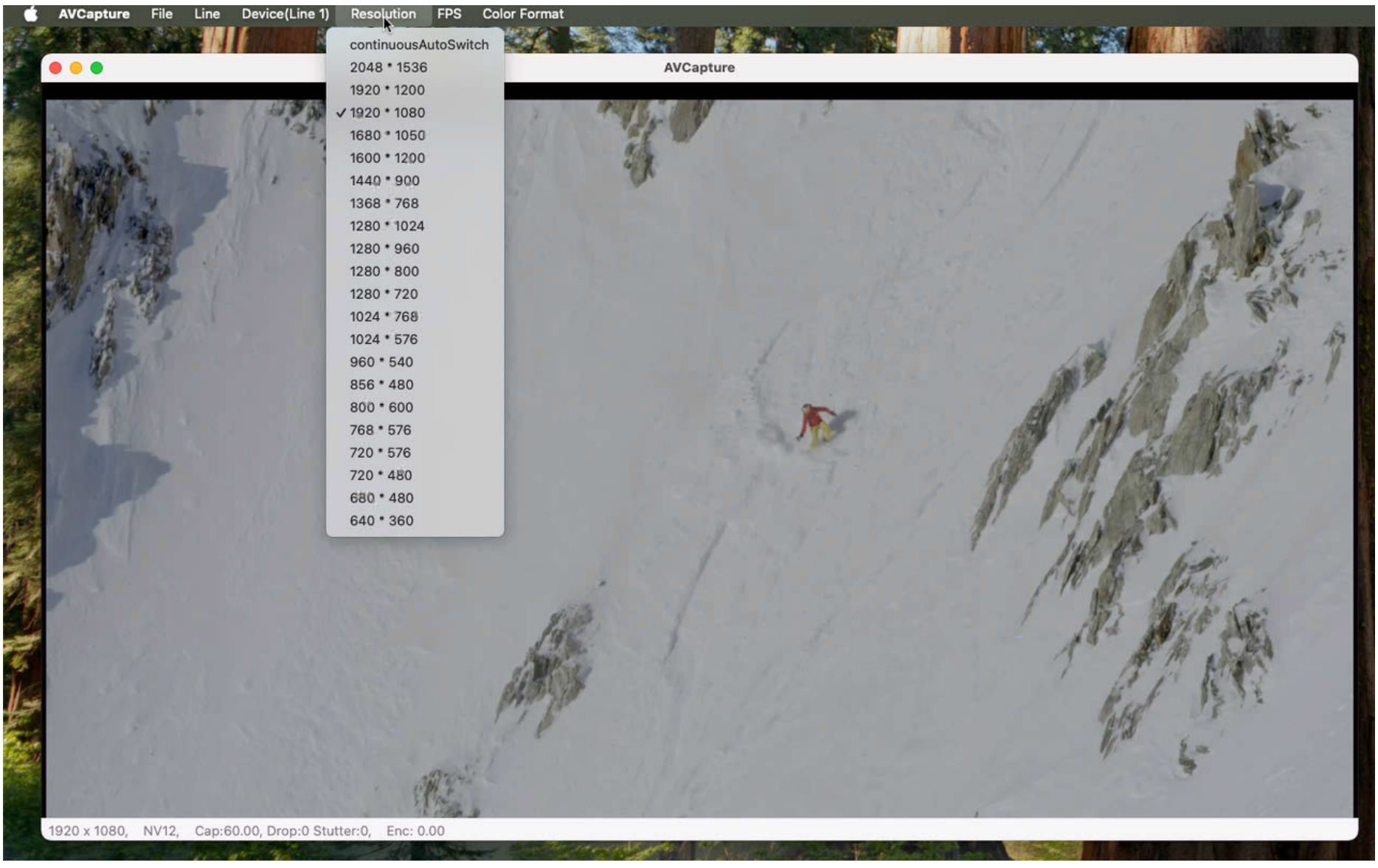
1. Start video capture: MMWStartVideoCapture
2. Create Event: MWCreateEvent
3. Register timer: MWRegisterTimer
4. Set time schedule: MWScheduleTimer
5. Wait for Event: MWTryWaitEvent, MWWaitEvent, MWMultiWaitEvent
6. Capture video data: MWCaptureVideoFrameToVirtualAddressEx
7. Get current capture status and release capture card resources: MWGetVideoCaptureStatus
8. Repeat steps 4, 5, 6, 7 until thread exits
9. Unregister timer: MWUnregisterTimer
10. Close Event: MWCloseEvent
11. Stop capture: MWStopVideoCapture

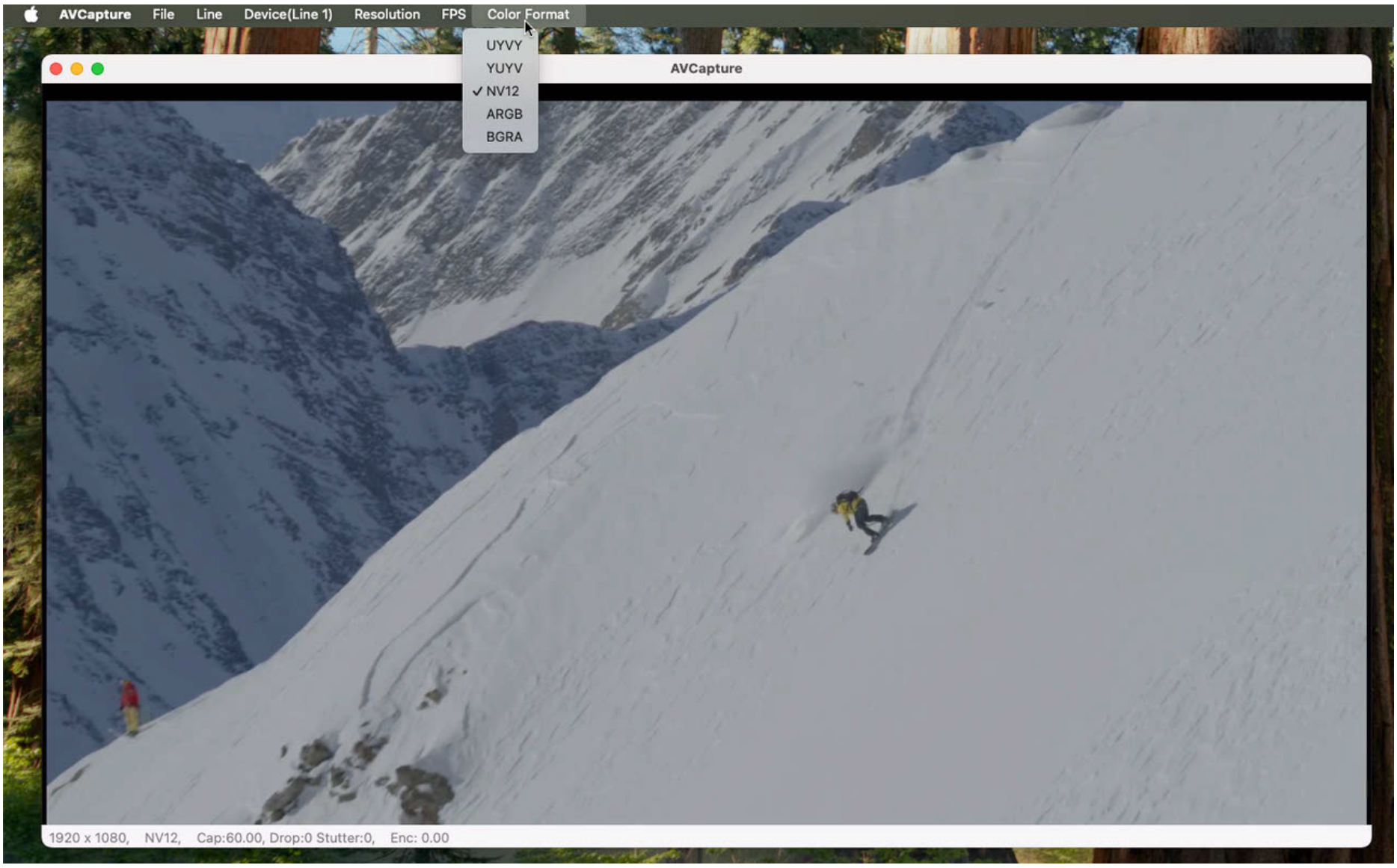
Audio Capture and Rendering Thread:

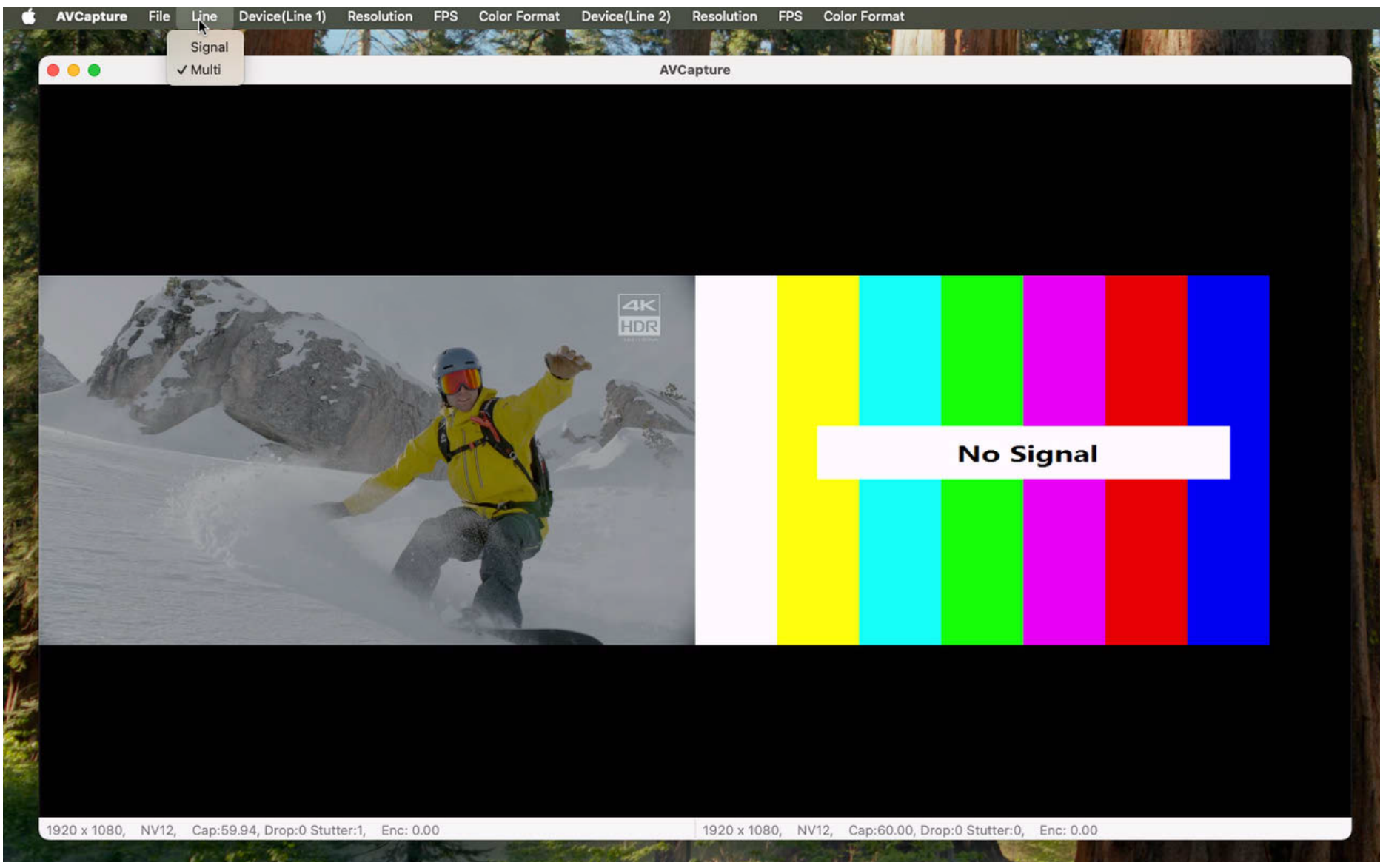
1. Start audio capture: MWStartAudioCapture
2. Create Event: MWCreateEvent
3. Register message notification: MWRegisterNotify
4. Wait for Event: MWTryWaitEvent, MWWaitEvent, MWMultiWaitEvent
5. Capture audio data: MWCaptureAudioFrame
6. Render audio data
7. Repeat steps 4, 5, 6 until thread exits
8. Unregister message notification: MWUnregisterNotify
9. Close Event: MWCloseEvent
10. Stop capture: MWStopAudioCapture

Result:

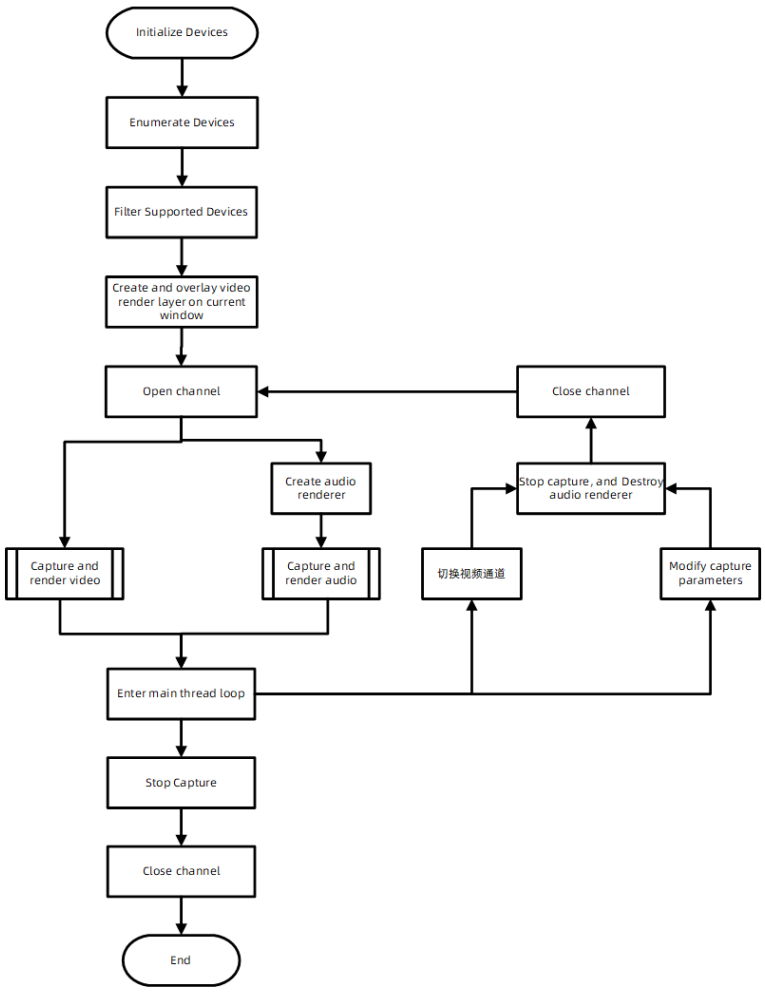

AVCapture File Line Device(Line 1) Resolution FPS Color Format
00-3 Pro Capture Quad HDMI
✓ 00-0 Pro Capture Quad HDMI
00-2 Pro Capture Quad HDMI
00-1 Pro Capture Quad HDMI
AVCapture
1920 x 1080, NV12, Cap:60.00, Drop:0 Stutter:0, Enc: 0.00

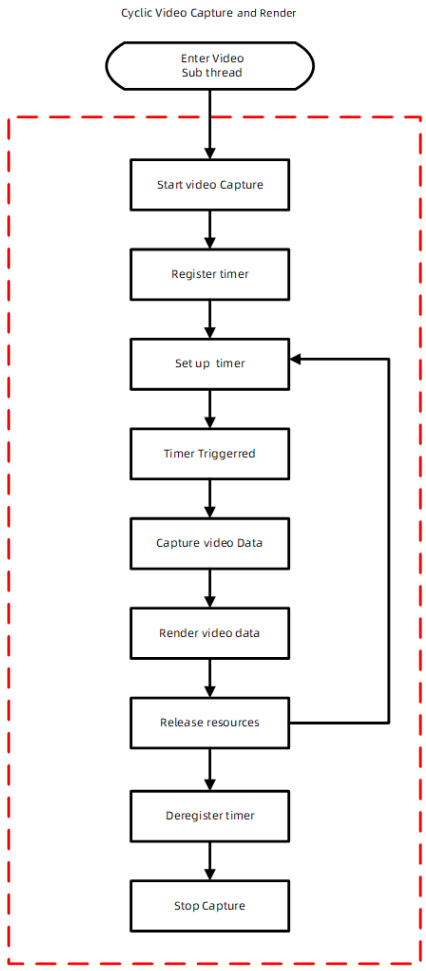

AVCapture File Line Device(Line 1) Resolution FPS Color Format
continuousAutoSwitch
2048 * 1536
1920 * 1200
✓ 1920 * 1080
1680 * 1050
1600 * 1200
1440 * 900
1368 * 768
1280 * 1024
1280 * 960
1280 * 800
1280 * 720
1024 * 768
1024 * 576
960 * 540
856 * 480
800 * 600
768 * 576
720 * 576
720 * 480
680 * 480
640 * 360
AVCapture
1920 x 1080, NV12, Cap:60.00, Drop:0 Stutter:0, Enc: 0.00

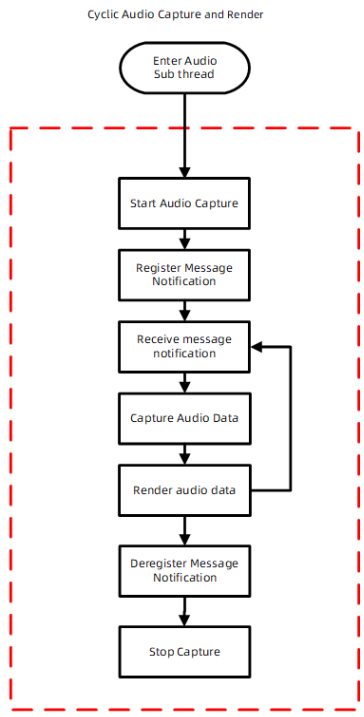

AVCapture File Line Device(Line 1) Resolution FPS Color Format
UYVY
YUYV
✓ NV12
ARGB
BGRA
AVCapture
1920 x 1080, NV12, Cap:60.00, Drop:0 Stutter:0, Enc: 0.00


AVCapture File Line Device(Line 1) Resolution FPS Color Format Device(Line 2) Resolution FPS Color Format
Signal
✓ Multi
AVCapture
4K
HDR
No Signal
1920 x 1080, NV12, Cap:59.94, Drop:0 Stutter:1, Enc: 0.00
1920 x 1080, NV12, Cap:60.00, Drop:0 Stutter:0, Enc: 0.00

Flowchart:


Initialize Devices
Enumerate Devices
Filter Supported Devices
Create and overlay video render layer on current window
Open channel
Close channel
Create audio renderer
Stop capture, and Destroy audio renderer
Capture and render video
Capture and render audio
切换视频通道
Modify capture parameters
Enter main thread loop
Stop Capture
Close channel
End


Cyclic Video Capture and Render
Enter Video Sub thread
Start video Capture
Register timer
Set up timer
Timer Triggerred
Capture video Data
Render video data
Release resources
Deregister timer
Stop Capture


Cyclic Audio Capture and Render
Enter Audio Sub thread
Start Audio Capture
Register Message Notification
Receive message notification
Capture Audio Data
Render audio data
Deregister Message Notification
Stop Capture

AVCapture2

Supported Hardware Devices: Pro Capture cards, USB Capture devices

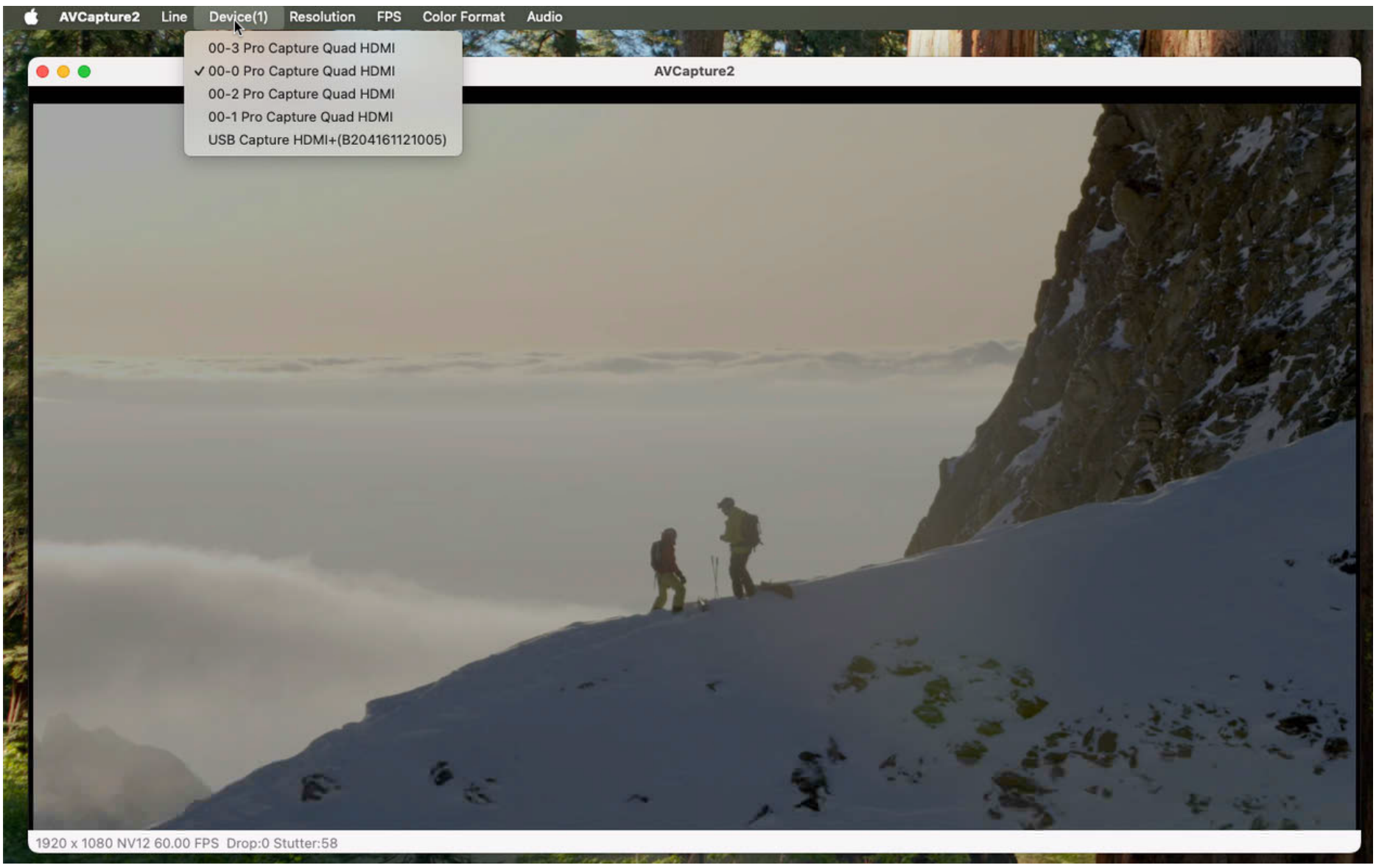
Contents Demonstrated in the Sample:

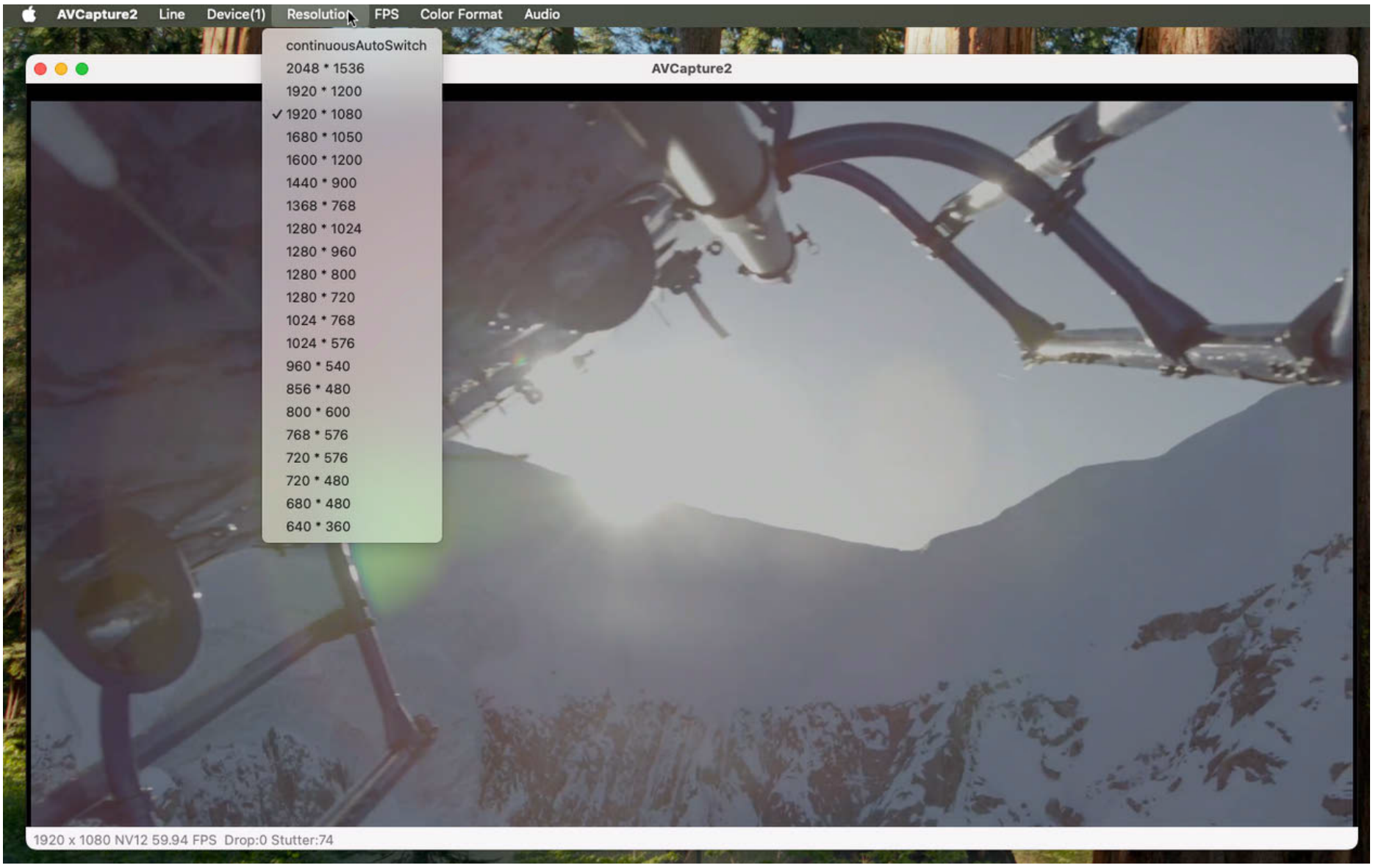
1. Preview captured video and monitor captured audio
2. Freely select video capture devices and their corresponding audio devices
3. Freely switch capture parameters: resolution, frame rate, color space
4. Support multi-channel capture: simultaneously capture one or two audio/video streams and render them separately

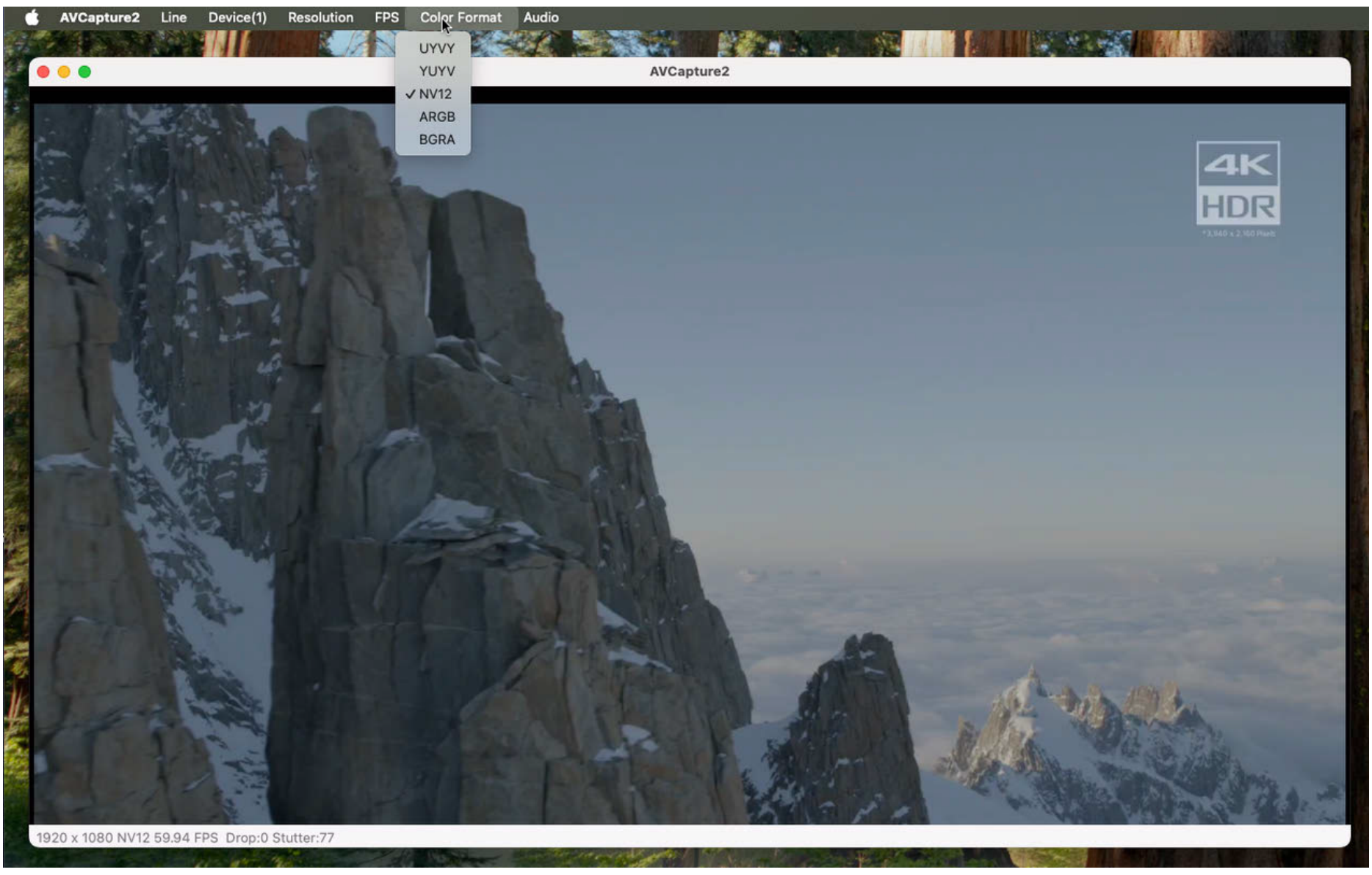
Call Logic:

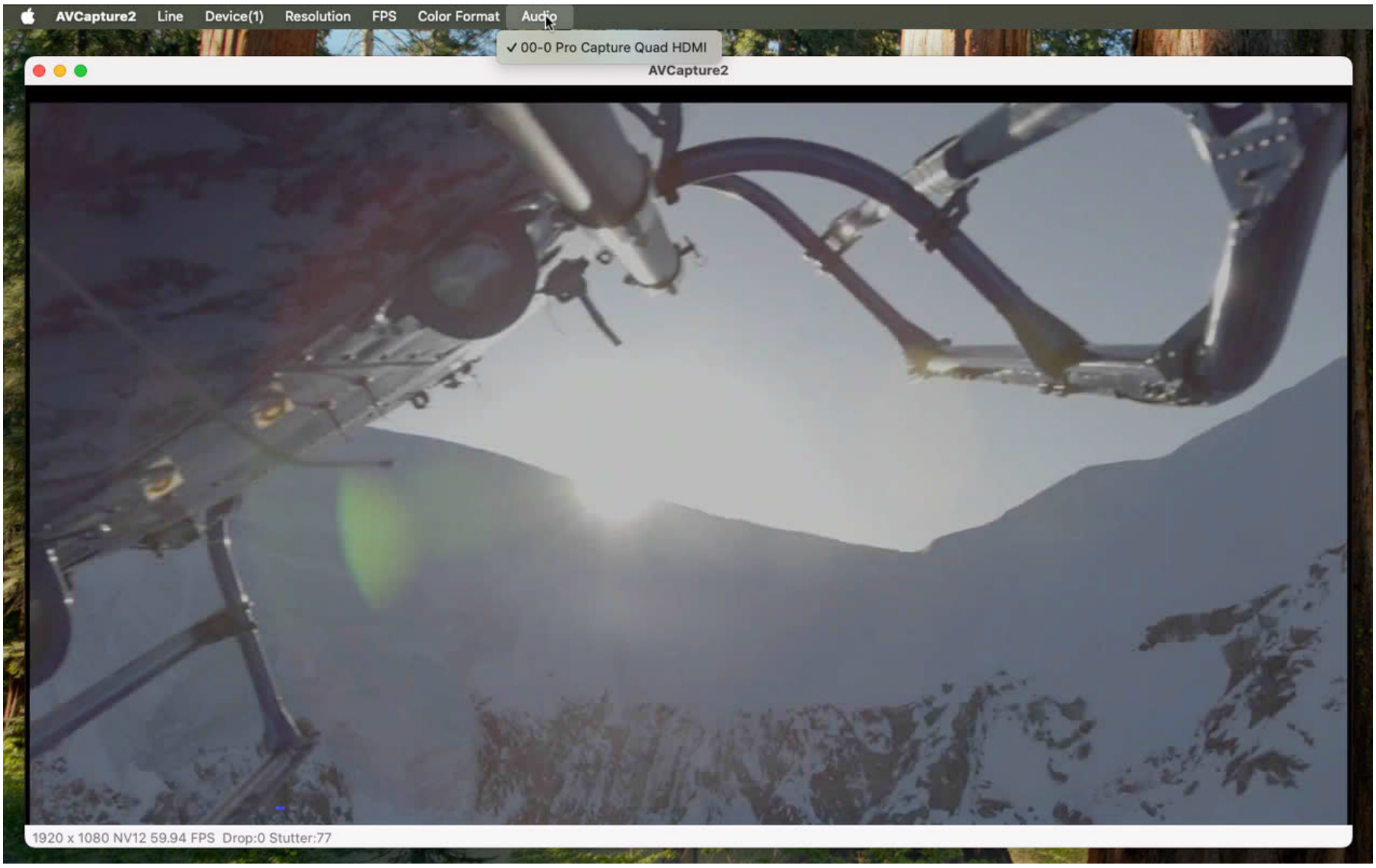
1. Obtain version information, initialize and enumerate devices: MWGetVersion, MWCaptureInitInstance, MWRefreshDevice, MWGetChannelCount
2. Filter supported devices based on device information and interface information: MWGetChannelInfoByIndex
3. Create a video rendering layer and overlay it on the current window: createPreviewLayer
4. Open channel: MWGetDevicePath, MWOpenChannelByPath
5. Create audio renderer: MWAudioRenderCreate, MWAudioRenderStart
6. Start video capture: MWCreateVideoCapture
7. Start audio capture: MWCreateAudioCapture
8. Enter main thread loop
9. Click menu to switch audio/video channels or capture parameters:
    i. Stop audio capture: MWDestroyAudioCapture
    ii. Stop video capture: MWDestroyVideoCapture
    iii. Destroy audio renderer: MWAudioRenderStop, MWAudioRenderDestroy
    iv. Close channel: MWCloseChannel
    v. Open channel and capture/render according to set channel and capture parameters: steps 4, 5, 6, 7
10. After clicking the close button, exit capture and release resources:
    i. Stop audio capture: MWDestroyAudioCapture
    ii. Stop video capture: MWDestroyVideoCapture
    iii. Destroy audio renderer: MWAudioRenderStop, MWAudioRenderDestroy
    iv. Close channel: MWCloseChannel
    v. Destroy instance: MWCaptureExitInstance.

Result:

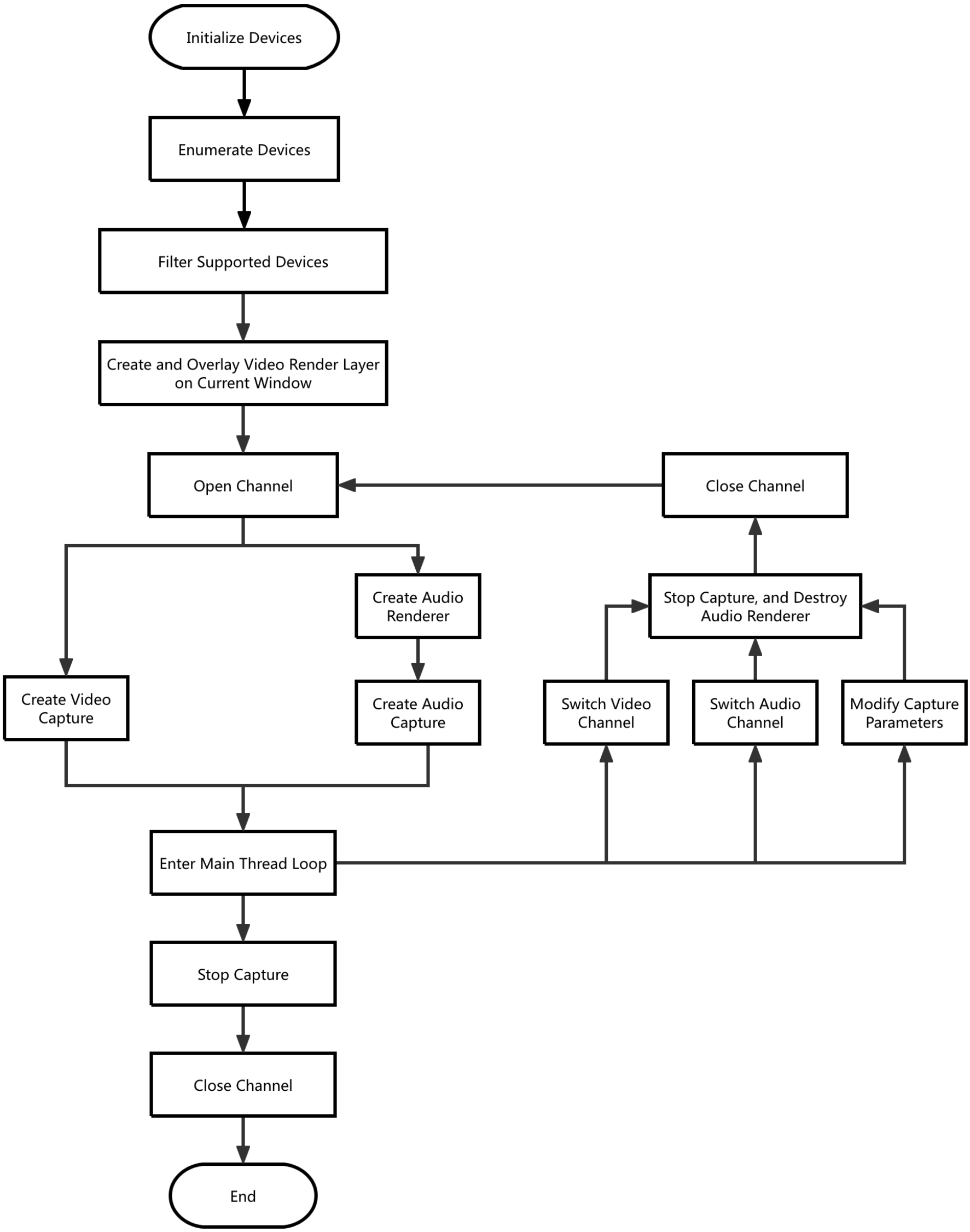



AVCapture2
Line
Device(1)
Resolution
FPS
Color Format
Audio
continuousAutoSwitch
2048 * 1536
1920 * 1200
✓ 1920 * 1080
1680 * 1050
1600 * 1200
1440 * 900
1368 * 768
1280 * 1024
1280 * 960
1280 * 800
1280 * 720
1024 * 768
1024 * 576
960 * 540
856 * 480
800 * 600
768 * 576
720 * 576
720 * 480
680 * 480
640 * 360
AVCapture2
1920 x 1080 NV12 59.94 FPS Drop:0 Stutter:74


AVCapture2
Line
Device(1)
Resolution
FPS
Color Format
Audio
UYVY
YUYV
✓ NV12
ARGB
BGRA
AVCapture2
4K
HDR
1920 x 1080 NV12 59.94 FPS Drop:0 Stutter:77


AVCapture2
Line
Device(1)
Resolution
FPS
Color Format
Audio
✓ 00-0 Pro Capture Quad HDMI
AVCapture2
1920 x 1080 NV12 59.94 FPS Drop:0 Stutter:77

Flowchart:


Initialize Devices
Enumerate Devices
Filter Supported Devices
Create and Overlay Video Render Layer on Current Window
Open Channel
Close Channel
Create Audio Renderer
Stop Capture, and Destroy Audio Renderer
Create Video Capture
Create Audio Capture
Switch Video Channel
Switch Audio Channel
Modify Capture Parameters
Enter Main Thread Loop
Stop Capture
Close Channel
End