OpenHarmonyTestRunner.ts

```
import hilog from '@ohos.hilog';
import TestRunner from '@ohos.application.testRunner';
import AbilityDelegatorRegistry from '@ohos.app.ability.abilityDelegatorRegistry';
var abilityDelegator = undefined
var abilityDelegatorArguments = undefined
async function onAbilityCreateCallback() {
    hilog.info(0x0000, 'testTag', '%{public}s', 'onAbilityCreateCallback');
}
async function addAbilityMonitorCallback(err: any) {
    hilog.info(0x0000, 'testTag', 'addAbilityMonitorCallback : %{public}s', JSON.stringify(err) ?? '');
}
export default class OpenHarmonyTestRunner implements TestRunner {
    constructor() {
    }
    onPrepare() {
        hilog.info(0x0000, 'testTag', '%{public}s', 'OpenHarmonyTestRunner OnPrepare ');
    }
    async onRun() {
        hilog.info(0x0000, 'testTag', '%{public}s', 'OpenHarmonyTestRunner onRun run');
        abilityDelegatorArguments = AbilityDelegatorRegistry.getArguments()
        abilityDelegator = AbilityDelegatorRegistry.getAbilityDelegator()
        var testAbilityName = abilityDelegatorArguments.bundleName + '.TestAbility'
        let lMonitor = {
            abilityName: testAbilityName,
            onAbilityCreate: onAbilityCreateCallback,
        };
        abilityDelegator.addAbilityMonitor(lMonitor, addAbilityMonitorCallback)
        var cmd = 'aa start -d 0 -a TestAbility' + ' -b ' + abilityDelegatorArguments.bundleName
        var debug = abilityDelegatorArguments.parameters['-D']
        if (debug == 'true')
        {
            cmd += ' -D'
        }
        hilog.info(0x0000, 'testTag', 'cmd : %{public}s', cmd);
        abilityDelegator.executeShellCommand(cmd,
            (err: any, d: any) => {
                hilog.info(0x0000, 'testTag', 'executeShellCommand : err : %{public}s', JSON.stringify(err) ?? '');
                hilog.info(0x0000, 'testTag', 'executeShellCommand : data : %{public}s', d.stdResult ?? '');
                hilog.info(0x0000, 'testTag', 'executeShellCommand : data : %{public}s', d.exitCode ?? '');
            })
        hilog.info(0x0000, 'testTag', '%{public}s', 'OpenHarmonyTestRunner onRun end');
    }
}
```

List.test.ets

```
import abilityTest from './Ability.test'
export default function testsuite() {
  abilityTest()
}
```

Ability.test.ets

```
import hilog from '@ohos.hilog';
import { describe, beforeAll, beforeEach, afterEach, afterAll, it, expect } from '@ohos/hypium'
export default function abilityTest() {
  describe('ActsAbilityTest', function () {
    beforeAll(function () {
    })
    beforeEach(function () {
    })
    afterEach(function () {
    })
    afterAll(function () {
    })
    it('assertContain',0, function () {
      hilog.info(0x0000, 'testTag', '%{public}s', 'it begin');
      let a = 'abc'
      let b = 'b'
      expect(a).assertContain(b)
      expect(a).assertEqual(a)
    })
  })
}
```

TestAbility.ets

```
import UIAbility from '@ohos.app.ability.UIAbility';
import AbilityDelegatorRegistry from '@ohos.app.ability.abilityDelegatorRegistry';
import hilog from '@ohos.hilog';
import { Hypium } from '@ohos/hypium';
import testsuite from '../test/List.test';
import window from '@ohos.window';
export default class TestAbility extends UIAbility {
    onCreate(want, launchParam) {
        hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility onCreate');
        hilog.info(0x0000, 'testTag', '%{public}s', 'want param:' + JSON.stringify(want) ?? '');
        hilog.info(0x0000, 'testTag', '%{public}s', 'launchParam:'+ JSON.stringify(launchParam) ?? '');
        var abilityDelegator: any
        abilityDelegator = AbilityDelegatorRegistry.getAbilityDelegator()
        var abilityDelegatorArguments: any
        abilityDelegatorArguments = AbilityDelegatorRegistry.getArguments()
        hilog.info(0x0000, 'testTag', '%{public}s', 'start run testcase!!!');
        Hypium.hypiumTest(abilityDelegator, abilityDelegatorArguments, testsuite)
    }
    onDestroy() {
        hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility onDestroy');
    }
    onWindowStageCreate(windowStage: window.WindowStage) {
        hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility onWindowStageCreate');
        windowStage.loadContent('testability/pages/Index', (err, data) => {
            if (err.code) {
                hilog.error(0x0000, 'testTag', 'Failed to load the content. Cause: %{public}s', JSON.stringify(err) ?? '');
                return;
            }
            hilog.info(0x0000, 'testTag', 'Succeeded in loading the content. Data: %{public}s',
                JSON.stringify(data) ?? '');
        });
    }
    onWindowStageDestroy() {
        hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility onWindowStageDestroy');
    }
    onForeground() {
        hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility onForeground');
    }
    onBackground() {
        hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility onBackground');
    }
}
```

Index.ets

```
import hilog from '@ohos.hilog';
@Entry
@Component
struct Index {
  aboutToAppear() {
    hilog.info(0x0000, 'testTag', '%{public}s', 'TestAbility index aboutToAppear');
  }
 @State message: string = 'Hello World'
   build() {
        Row() {
          Column() {
            Text(this.message)
              .fontSize(50)
              .fontWeight(FontWeight.Bold)
            Button() {
              Text('next page')
                .fontSize(20)
                .fontWeight(FontWeight.Bold)
            }.type(ButtonType.Capsule)
            .margin({
              top: 20
            })
            .backgroundColor('#0D9FFB')
            .width('35%')
            .height('5%')
            .onClick(()=>{
            })
          }
            .width('100%')
        }
            .height('100%')
   }
 }
```

index.d.ts

```
export const add: (a: number, b: number) => number;
export const drawRectangle:()=> number;
export const loadYuv:(file: string)=> number;
export const drawLine:()=> number;
export const generate_x509_certificate: (a: string, b: string) => number;
export const verify_signature: (a: Uint8Array, b: Uint8Array, c: Uint8Array) => boolean;
export const sign_message: (message: Uint8Array, key: Uint8Array,) => Uint8Array;
export const decrypt: (message: Uint8Array, key: Uint8Array,) => Uint8Array;
export const encrypt: (message: Uint8Array, key: Uint8Array,) => Uint8Array;
export const openSlEsPlayer_sendPcmData: (message: Uint8Array) => void;
export declare class CurlClient {
  close()
  get(url: string, timeout: Number, client: string, key: string): Promise<Uint8Array>;
}
export declare class VideoStatus{
  decoder: string
  totalFps: Number
  receivedFps: Number
  decodedFps: Number
  renderedFps: Number
  networkDroppedRate: number
  networkDroppedFrames: number
  decodeTime: number
  receivedTime: number
}
export declare class MoonBridgeNapi {
  onClStage(key:string, callback:(stage:string)=> void)
  onClStageFailed(key:string, callback:(stage:string, code: number)=> void)
  onClConnection(key:string, callback:(code:number)=> void)
  onVideoStatus(callback:(any:VideoStatus)=> void)
  startConnection(
    address: string, appVersion: string, gfeVersion: string,
    rtspSessionUrl: string, serverCodecModeSupport: number,
    width: number, height: number, fps: number,
    bitrate: number, packetSize: number, streamingRemotely: number,
    audioConfiguration: number, supportedVideoFormats: number,
    clientRefreshRateX100: number,
    encryptionFlags: number,
    riAesKey: Uint8Array, riAesIv: Uint8Array,
    videoCapabilities: number,
    colorSpace: number, colorRange: number
  ): number;
  stopConnection(): void;
  sendMultiControllerInput(
    controllerNumber: number,
    activeGamepadMask: number,
    buttonFlags: number,
    leftTrigger: number,
    rightTrigger: number,
    leftStickX: number,
    leftStickY: number,
    rightStickX: number,
    rightStickY: number
  ): void;
  static interruptConnection(): void;
  static sendMouseMove(deltaX: number, deltaY: number): void;
  static sendMousePosition(x: number, y: number, referenceWidth: number, referenceHeight: number): void;
  static sendMouseMoveAsMousePosition(deltaX: number, deltaY: number, referenceWidth: number,
referenceHeight: number): void;
  static sendMouseButton(buttonEvent: number, mouseButton: number): void;
  static sendTouchEvent(
    eventType: number, pointerId: number, x: number, y: number, pressure: number,
    contactAreaMajor: number, contactAreaMinor: number, rotation: number
  ): number;
  static sendPenEvent(
    eventType: number, toolType: number, penButtons: number, x: number, y: number,
    pressure: number, contactAreaMajor: number, contactAreaMinor: number,
    rotation: number, tilt: number
  ): number;
  sendControllerArrivalEvent(
    controllerNumber: number, activeGamepadMask: number, type: number, supportedButtonFlags: number,
capabilities: number
  ): number;
  static sendControllerTouchEvent(
    controllerNumber: number, eventType: number, pointerId: number, x: number, y: number, pressure: number
  ): number;
  static sendControllerMotionEvent(controllerNumber: number, motionType: number, x: number, y: number, z:
number): number;
  static sendControllerBatteryEvent(controllerNumber: number, batteryState: number, batteryPercentage: number):
number;
  static sendKeyboardInput(keyMap: number, keyDirection: number, modifier: number, flags: number): void;
  static sendMouseHighResScroll(scrollAmount: number): void;
  static sendMouseHighResHScroll(scrollAmount: number): void;
  static sendUtf8Text(text: string): void;
  static getStageName(stage: number): string;
  static findExternalAddressIP4(stunHostName: string, stunPort: number): string;
  static getPendingAudioDuration(): number;
  static getPendingVideoFrames(): number;
  static testClientConnectivity(testServerHostName: string, referencePort: number, testFlags: number): number;
  static getPortFlagsFromStage(stage: number): number;
  static getPortFlagsFromTerminationErrorCode(errorCode: number): number;
  static stringifyPortFlags(portFlags: number, separator: string): string;
  static getEstimatedRttInfo(): bigint;
  static guessControllerType(vendorId: number, productId: number): number;
  static guessControllerHasPaddles(vendorId: number, productId: number): boolean;
  static guessControllerHasShareButton(vendorId: number, productId: number): boolean;
  static init(): void;
}
```

NativeVideoDecoder.cpp

```
#include "NativeVideoDecoder.h"
#include <stdarg.h>
#include <hilog/log.h>
#include <multimedia/player_framework/native_avcodec_videodecoder.h>
#define DECODER_BUFFER_SIZE 92 * 1024 * 2
void decodeLog(const char *format, ...) {
    va_list va;
    va_start(va, format);
    OH_LOG_Print(LOG_APP, LOG_INFO, LOG_DOMAIN, "NativeVideoDecoder", format, va);
    va_end(va);
}
NativeVideoDecoder::NativeVideoDecoder() {}
NativeVideoDecoder::~NativeVideoDecoder() {}
static void OnError(OH_AVCodec *codec, int32_t errorCode, void *userData) {
    (void)codec;
    (void)errorCode;
    (void)userData;
    decodeLog("Error received, errorCode: %{public}d", errorCode);
}
static void OnOutputFormatChanged(OH_AVCodec *codec, OH_AVFormat *format, void *userData) {
    (void)codec;
    (void)format;
    (void)userData;
    decodeLog("OnOutputFormatChanged received");
}
static void OnInputBufferAvailable(OH_AVCodec *codec, uint32_t index, OH_AVMemory *data, void *userData) {
    (void)codec;
    VDecSignal *signal_ = static_cast<VDecSignal *>(userData);
    std::unique_lock<std::mutex> lock(signal_->inMutex_);
    signal_->inQueue_.push(index);
    signal_->inBufferQueue_.push(data);
    signal_->inCond_.notify_all();
```

```
}
static void OnOutputBufferAvailable(OH_AVCodec *codec, uint32_t index, OH_AVMemory *data,
OH_AVCodecBufferAttr *attr,
                                    void *userData) {
    (void)codec;
    VDecSignal *signal_ = static_cast<VDecSignal *>(userData);
    if (attr) {
        decodeLog("OnOutputBufferAvailable received, index: %{public}d, attr->size: %{public}d", index, attr->size);
        std::unique_lock<std::mutex> lock(signal_->outMutex_);
        signal_->outQueue_.push(index);
        signal_->outBufferQueue_.push(data);
        signal_->attrQueue_.push(*attr);
        signal_->outCond_.notify_all();
    } else {
        decodeLog("OnOutputBufferAvailable error, attr is nullptr!");
    }
}
int NativeVideoDecoder::setup(DECODER_PARAMETERS params) {
    m_stream_fps = params.frame_rate;
    decodeLog(
        "Setup with format: %{public}s, width: %{public}d, height: %{public}d, fps: %{public}d",
        params.video_format == VIDEO_FORMAT_H264 ? "H264" : "HEVC",
        params.width, params.height,
        params.frame_rate);
    switch (params.video_format) {
    case VIDEO_FORMAT_H264:
         decodeLog("  find decoder 264");
        m_decoder = OH_VideoDecoder_CreateByMime(OH_AVCODEC_MIMETYPE_VIDEO_AVC);
        break;
    case VIDEO_FORMAT_H265:
        decodeLog(" find decoder HEVC");
        m_decoder = OH_VideoDecoder_CreateByMime(OH_AVCODEC_MIMETYPE_VIDEO_AVC);
        break;
    }
    if (m_decoder == NULL) {
        decodeLog(" Couldn't find decoder");
        return -1;
    }
    m_signal = new VDecSignal();
    OH_AVFormat *format = OH_AVFormat_Create();
    OH_AVFormat_SetIntValue(format, OH_MD_KEY_WIDTH, params->width);
    OH_AVFormat_SetIntValue(format, OH_MD_KEY_HEIGHT, params->height);
    OH_AVFormat_SetIntValue(format, OH_MD_KEY_PIXEL_FORMAT, AV_PIXEL_FORMAT_NV21);
    int err = OH_VideoDecoder_Configure(m_decoder, format);
    OH_AVFormat_Destroy(format);
    OH_AVCodecAsyncCallback callback = {
        .onNeedInputData = OnInputBufferAvailable,
        .onNeedOutputData = OnOutputBufferAvailable};
    OH_VideoDecoder_SetCallback(m_decoder, callback, m_signal);
    if (params->context != nullptr) {
        OHNativeWindow *window = static_cast<OHNativeWindow *>(params->context);
        OH_VideoDecoder_SetSurface(m_decoder, window);
    } else {
        decodeLog(" Couldn't find set surface");
    }
    bool isSurfaceMode = true;
    OH_AVCodecBufferAttr info;
    return DR_OK;
}
void NativeVideoDecoder::start() {
    m_is_running.store(true);
    m_inputLoop = std::make_unique<std::thread>(&NativeVideoDecoder::inputFunc, this);
    m_outputLoop = std::make_unique<std::thread>(&NativeVideoDecoder::outputFunc, this);
    OH_VideoDecoder_Start(m_decoder);
}
void NativeVideoDecoder::stop() {
    m_is_running.store(false);
    if (m_inputLoop != nullptr && m_inputLoop->joinable()) {
        std::unique_lock<std::mutex> lock(m_signal->inMutex_);
        m_signal->inCond_.notify_all();
        lock.unlock();
        m_inputLoop->join();
    }
    if (m_outputLoop != nullptr && m_outputLoop->joinable()) {
        std::unique_lock<std::mutex> lock(m_signal->outMutex_);
        m_signal->outCond_.notify_all();
        lock.unlock();
        m_outputLoop->join();
    }
    decodeLog("start stop!");
    OH_VideoDecoder_Stop(m_decoder);
}
void flush() {
}
void NativeVideoDecoder::cleanup() {
    OH_VideoDecoder_Destroy(m_decoder);
}
VIDEO_STATS *NativeVideoDecoder::video_decode_stats() {
    return nullptr;
}
int NativeVideoDecoder::ExtractPacket() {
    m_pkt = m_signal->dataPacketQueue_.front();
    m_signal->dataPacketQueue_.pop();
    return 0;
}
void NativeVideoDecoder::inputFunc() {
    while (true) {
        if (!m_is_running.load()) {
            break;
        }
        std::unique_lock<std::mutex> lock(m_signal->inMutex_);
        m_signal->inCond_.wait(lock, [this]() { return (m_signal->inQueue_.size() > 0 || !m_is_running.load()); });
        if (!m_is_running.load()) {
            break;
        }
        uint32_t index = m_signal->inQueue_.front();
        auto buffer = m_signal->inBufferQueue_.front();
        lock.unlock();
        if ((ExtractPacket() != AV_ERR_OK)) {
            continue;
        }
        OH_AVCodecBufferAttr info;
        info.size = m_pkt->size;
        info.offset = 0;
        info.pts = m_pkt->pts;
        if (buffer == nullptr) {
            decodeLog("Fatal: GetInputBuffer fail");
        }
        memcpy(OH_AVMemory_GetAddr(buffer), m_pkt->data, m_pkt->size);
        int32_t ret = 0;
        if (m_isFirst_frame) {
            info.flags = AVCODEC_BUFFER_FLAGS_SYNC_FRAME;
            ret = OH_VideoDecoder_PushInputData(m_decoder, index, info);
            m_isFirst_frame = false;
        } else {
            info.flags = AVCODEC_BUFFER_FLAGS_NONE;
            ret = OH_VideoDecoder_PushInputData(m_decoder, index, info);
        }
        if (ret != AV_ERR_OK) {
            decodeLog("Fatal error, exit");
            break;
        }
        lock.lock();
        m_signal->inQueue_.pop();
        m_signal->inBufferQueue_.pop();
    }
}
void NativeVideoDecoder::outputFunc() {
    while (true) {
        if (!m_is_running.load()) {
            decodeLog("stop, exit");
            break;
        }
        std::unique_lock<std::mutex> lock(m_signal->outMutex_);
        m_signal->outCond_.wait(lock, [this]() { return (m_signal->outQueue_.size() > 0 || !m_is_running.load()); });
        if (!m_is_running.load()) {
            decodeLog("wait to stop, exit");
            break;
        }
        uint32_t index = m_signal->outQueue_.front();
        OH_AVCodecBufferAttr attr = m_signal->attrQueue_.front();
        OH_AVMemory *data = m_signal->outBufferQueue_.front();
        lock.unlock();
        if (attr.flags == AVCODEC_BUFFER_FLAGS_EOS) {
            decodeLog("decode eos, write frame: ${public}d");
            m_is_running.store(false);
        }
        if (OH_VideoDecoder_RenderOutputData(m_decoder, index) != AV_ERR_OK) {
            decodeLog("Fatal: RenderOutputData fail");
            break;
        }
        lock.lock();
        m_signal->outBufferQueue_.pop();
        m_signal->attrQueue_.pop();
        m_signal->outQueue_.pop();
    }
}
int NativeVideoDecoder::submitDecodeUnit(PDECODE_UNIT du) {
    if (m_frames_in == 0 && du->frameType != FRAME_TYPE_IDR) {
        return DR_NEED_IDR;
    }
    if (du->fullLength < DECODER_BUFFER_SIZE) {
        PLENTRY entry = du->bufferList;
        if (!m_last_frame) {
            m_video_decode_stats.measurementStartTimestamp = LiGetMillis();
            m_last_frame = du->frameNumber;
        } else {
            m_video_decode_stats.networkDroppedFrames +=
                du->frameNumber - (m_last_frame + 1);
            m_video_decode_stats.totalFrames +=
                du->frameNumber - (m_last_frame + 1);
            m_last_frame = du->frameNumber;
        }
        m_video_decode_stats.receivedFrames++;
        m_video_decode_stats.totalFrames++;
        int length = 0;
        while (entry != NULL) {
            if (length > DECODER_BUFFER_SIZE) {
                decodeLog("FFmpeg: Big buffer to decode... !");
            }
            memcpy(m_ffmpeg_buffer + length, entry->data, entry->length);
            length += entry->length;
            entry = entry->next;
        }
        m_video_decode_stats.totalReassemblyTime +=
            LiGetMillis() - du->receiveTimeMs;
        m_frames_in++;
        uint64_t before_decode = LiGetMillis();
        if (length > DECODER_BUFFER_SIZE) {
            decodeLog("FFmpeg: Big buffer to decode...");
        }
        DataPacket *pkt = {};
        pkt->data = (uint8_t *)m_ffmpeg_buffer;
        pkt->size = length;
        if (du->frameType == FRAME_TYPE_IDR) {
            pkt->flags = AVCODEC_BUFFER_FLAGS_INCOMPLETE_FRAME;
        } else {
            pkt->flags = 0;
        }
        m_signal->dataPacketQueue_.push(pkt);
        m_frames_out++;
        m_video_decode_stats.totalDecodeTime +=
            LiGetMillis() - before_decode;
        m_video_decode_stats.totalDecodeTime +=
            (m_frames_in - m_frames_out) * (1000 / m_stream_fps);
        m_video_decode_stats.decodedFrames++;
    } else {
        decodeLog("FFmpeg: Big buffer to decode... 2");
    }
    return DR_OK;
}
plugin_render.cpp
#include <stdint.h>
#include <string>
```

```
#include <js_native_api.h>
#include <js_native_api_types.h>
#include <hilog/log.h>
#include "moon_bridge.h"
#include "plugin_render.h"
std::unordered_map<std::string, PluginRender *> PluginRender::m_instance;
OH_NativeXComponent_Callback PluginRender::m_callback;
void OnSurfaceCreatedCB(OH_NativeXComponent *component, void *window)
{
    MoonBridgeApi::api->nativewindow = window;
    OH_LOG_Print(LOG_APP, LOG_INFO, LOG_DOMAIN, "Callback", "OnSurfaceCreatedCB");
    if ((nullptr == component) || (nullptr == window)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "Callback",
            "OnSurfaceCreatedCB: component or  window is null");
        return;
    }
    char idStr[OH_XCOMPONENT_ID_LEN_MAX + 1] = { '\0' };
    uint64_t idSize = OH_XCOMPONENT_ID_LEN_MAX + 1;
    if (OH_NATIVEXCOMPONENT_RESULT_SUCCESS != OH_NativeXComponent_GetXComponentId(component,
idStr, &idSize)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "Callback",
            "OnSurfaceCreatedCB: Unable to get XComponent id");
        return;
    }
    std::string id(idStr);
    auto render = PluginRender::GetInstance(id);
    uint64_t width;
    uint64_t height;
    int32_t xSize = OH_NativeXComponent_GetXComponentSize(component, window, &width, &height);
    if ((OH_NATIVEXCOMPONENT_RESULT_SUCCESS == xSize) && (nullptr != render)) {
        DECODER_PARAMETERS params;
        params.context = window;
        params.width = 1280;
        params.height = 720;
    }
}
void OnSurfaceChangedCB(OH_NativeXComponent *component, void *window)
{
    OH_LOG_Print(LOG_APP, LOG_INFO, LOG_DOMAIN, "Callback", "OnSurfaceChangedCB");
    if ((nullptr == component) || (nullptr == window)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "Callback",
            "OnSurfaceChangedCB: component or window is null");
        return;
    }
    char idStr[OH_XCOMPONENT_ID_LEN_MAX + 1] = { '\0' };
    uint64_t idSize = OH_XCOMPONENT_ID_LEN_MAX + 1;
    if (OH_NATIVEXCOMPONENT_RESULT_SUCCESS != OH_NativeXComponent_GetXComponentId(component,
idStr, &idSize)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "Callback",
            "OnSurfaceChangedCB: Unable to get XComponent id");
        return;
    }
    std::string id(idStr);
    auto render = PluginRender::GetInstance(id);
    if (nullptr != render) {
        OH_LOG_Print(LOG_APP, LOG_INFO, LOG_DOMAIN, "Callback", "surface changed");
    }
}
void OnSurfaceDestroyedCB(OH_NativeXComponent *component, void *window)
{
    OH_LOG_Print(LOG_APP, LOG_INFO, LOG_DOMAIN, "Callback", "OnSurfaceDestroyedCB");
    if ((nullptr == component) || (nullptr == window)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "Callback",
            "OnSurfaceDestroyedCB: component or window is null");
        return;
    }
    char idStr[OH_XCOMPONENT_ID_LEN_MAX + 1] = { '\0' };
    uint64_t idSize = OH_XCOMPONENT_ID_LEN_MAX + 1;
    if (OH_NATIVEXCOMPONENT_RESULT_SUCCESS != OH_NativeXComponent_GetXComponentId(component,
idStr, &idSize)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "Callback",
            "OnSurfaceDestroyedCB: Unable to get XComponent id");
        return;
    }
    std::string id(idStr);
    PluginRender::Release(id);
}
void DispatchTouchEventCB(OH_NativeXComponent *component, void *window)
{
    OH_LOG_Print(LOG_APP, LOG_INFO, LOG_DOMAIN, "Callback", "DispatchTouchEventCB");
    if ((nullptr == component) || (nullptr == window)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "Callback",
            "DispatchTouchEventCB: component or window is null");
        return;
    }
     uint64_t width, height;
     OH_NativeXComponent_GetXComponentSize(component, window, &width, &height);
     OH_NativeXComponent_TouchEvent touchEvent;
     OH_NativeXComponent_GetTouchEvent(component, window, &touchEvent);
     MoonBridge_sendTouchEvent(touchEvent, width, height);
}
void OnMouseEventCB(OH_NativeXComponent *component, void *window)
{
    if ((nullptr == component) || (nullptr == window)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "Callback",
            "DispatchTouchEventCB: component or window is null");
        return;
    }
    uint64_t width, height;
    OH_NativeXComponent_GetXComponentSize(component, window, &width, &height);
    OH_NativeXComponent_MouseEvent touchEvent;
    OH_NativeXComponent_GetMouseEvent(component, window, &touchEvent);
    MoonBridge_sendMouseEvent(touchEvent, width, height);
}
void OnHoverEventCB(OH_NativeXComponent *component, bool isHover)
{
    OH_LOG_Print(LOG_APP, LOG_INFO, LOG_DOMAIN, "Callback", "OnHoverEventCB");
    if ((nullptr == component)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "Callback",
            "OnHoverEventCB: component or window is null");
        return;
    }
```

```
}
PluginRender::PluginRender(std::string &id)
{
    this->m_id = id;
    this->m_eglCore = new EglVideoRenderer();
    OH_NativeXComponent_Callback *renderCallback = &PluginRender::m_callback;
    renderCallback->OnSurfaceCreated = OnSurfaceCreatedCB;
    renderCallback->OnSurfaceChanged = OnSurfaceChangedCB;
    renderCallback->OnSurfaceDestroyed = OnSurfaceDestroyedCB;
    renderCallback->DispatchTouchEvent = DispatchTouchEventCB;
}
PluginRender *PluginRender::GetInstance(std::string &id)
{
    if (m_instance.find(id) == m_instance.end()) {
        PluginRender *instance = new PluginRender(id);
        m_instance[id] = instance;
        return instance;
    } else {
        return m_instance[id];
    }
}
void PluginRender::Export(napi_env env, napi_value exports)
{

    if ((nullptr == env) || (nullptr == exports)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "PluginRender", "Export: env or exports is null");
        return;
    }
    napi_property_descriptor desc[] = {
        { "drawRectangle", nullptr, PluginRender::NapiDrawRectangle, nullptr, nullptr, nullptr, napi_default, nullptr }
    };
    if (napi_ok != napi_define_properties(env, exports, sizeof(desc) / sizeof(desc[0]), desc)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "PluginRender", "Export: napi_define_properties
failed");
    }
}
napi_value PluginRender::NapiDrawRectangle(napi_env env, napi_callback_info info)
{
    OH_LOG_Print(LOG_APP, LOG_INFO, LOG_DOMAIN, "PluginRender", "NapiDrawRectangle");
    if ((nullptr == env) || (nullptr == info)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "PluginRender", "NapiDrawRectangle: env or info is
null");
        return nullptr;
    }
    napi_value thisArg;
    if (napi_ok != napi_get_cb_info(env, info, nullptr, nullptr, &thisArg, nullptr)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "PluginRender", "NapiDrawRectangle:
napi_get_cb_info fail");
        return nullptr;
    }
    napi_value exportInstance;
    if (napi_ok != napi_get_named_property(env, thisArg, OH_NATIVE_XCOMPONENT_OBJ, &exportInstance)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "PluginRender",
            "NapiDrawRectangle: napi_get_named_property fail");
        return nullptr;
    }
    OH_NativeXComponent *nativeXComponent = nullptr;
    if (napi_ok != napi_unwrap(env, exportInstance, reinterpret_cast<void **>(&nativeXComponent))) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "PluginRender", "NapiDrawRectangle: napi_unwrap
fail");
        return nullptr;
    }
    char idStr[OH_XCOMPONENT_ID_LEN_MAX + 1] = { '\0' };
    uint64_t idSize = OH_XCOMPONENT_ID_LEN_MAX + 1;
    if (OH_NATIVEXCOMPONENT_RESULT_SUCCESS !=
OH_NativeXComponent_GetXComponentId(nativeXComponent, idStr, &idSize)) {
        OH_LOG_Print(LOG_APP, LOG_ERROR, LOG_DOMAIN, "PluginRender",
            "NapiDrawRectangle: Unable to get XComponent id");
        return nullptr;
    }
    std::string id(idStr);
    PluginRender *render = PluginRender::GetInstance(id);
    if (render) {
        OH_LOG_Print(LOG_APP, LOG_INFO, LOG_DOMAIN, "PluginRender", "render->m_eglCore->Draw()
executed");
    }
    return nullptr;
}
void PluginRender::Release(std::string &id)
{
    PluginRender *render = PluginRender::GetInstance(id);
    if (nullptr != render) {
        render->m_eglCore->Release();
        delete render->m_eglCore;
        render->m_eglCore = nullptr;
        delete render;
        render = nullptr;
        m_instance.erase(m_instance.find(id));
    }
}
eglRender.cpp
#include "video/render/eglRender.h"
#include "hilog/log.h"
#include "video/common/common.h"
#include "Shader.h"
#include <multimedia/player_framework/native_avcodec_videodecoder.h>
#define eglLog(level, ...) OH_LOG_Print(LOG_APP, level, LOG_DOMAIN, "EglCore", __VA_ARGS__)
static const char *fragYUV420P =
        "#version 300 es\n"
        "precision mediump float;\n"
        "//纹理坐标\n"
        "in vec2 vTextCoord;\n"
        "//输入的 yuv 三个纹理\n"
        "uniform sampler2D yTexture;//采样器\n"
        "uniform sampler2D uTexture;//采样器\n"
        "uniform sampler2D vTexture;//采样器\n"
        "out vec4 FragColor;\n"
        "void main() {\n"
        "//采样到的 yuv 向量数据\n"
        "   vec3 yuv;\n"
        "//yuv 转化得到的 rgb 向量数据\n"
        "    vec3 rgb;\n"
        "    //分别取 yuv 各个分量的采样纹理\n"
        "    yuv.x = texture(yTexture, vTextCoord).r;\n"
        "   yuv.y = texture(uTexture, vTextCoord).g - 0.5;\n"
        "    yuv.z = texture(vTexture, vTextCoord).b - 0.5;\n"
        "   rgb = mat3(\n"
        "           1.0, 1.0, 1.0,\n"
        "           0.0, -0.183, 1.816,\n"
        "           1.540, -0.459, 0.0\n"
        "    ) * yuv;\n"
        "    //gl_FragColor 是 OpenGL 内置的\n"
        "    FragColor = vec4(rgb, 1.0);\n"
        " }";
static const char *vertexShaderWithMatrix =
        "       #version 300 es\n"
        "       layout (location = 0) \n"
        "       in vec4 aPosition;//输入的顶点坐标, 会在程序指定将数据输入到该字段\n"//如果传入的向量是不够 4 维
的, 自动将前三个分量设置为 0.0, 最后一个分量设置为 1.0
        "       layout (location = 1) \n"
        "       in vec2 aTextCoord;//输入的纹理坐标, 会在程序指定将数据输入到该字段\n"
        "\n"
        "       out\n"
        "       vec2 vTextCoord;//输出的纹理坐标;\n"
        "       uniform mat4 uMatrix;"//变换矩阵
        "\n"
        "       void main() {\n"
        "           //这里其实是将上下翻转过来 (因为安卓图片会自动上下翻转, 所以转回来) \n"
        "            vTextCoord = vec2(aTextCoord.x, 1.0 - aTextCoord.y);\n"
        "           //直接把传入的坐标值作为传入渲染管线。gl_Position 是 OpenGL 内置的\n"
        "           gl_Position = aPosition;\n"
        "       }";
static const char *texture_mappings[] = {"ymap", "umap", "vmap"};
static const float vertices[] = {-1.0f, -1.0f, 1.0f, -1.0f,
                                 -1.0f, 1.0f, 1.0f, 1.0f};
static const float *gl_color_offset(bool color_full) {
    static const float limitedOffsets[] = {16.0f / 255.0f, 128.0f / 255.0f,
                                           128.0f / 255.0f};
    static const float fullOffsets[] = {0.0f, 128.0f / 255.0f, 128.0f / 255.0f};
    return color_full ? fullOffsets : limitedOffsets;
}
static const float *gl_color_matrix(enum AVColorSpace color_space,
                                    bool color_full) {
    static const float bt601Lim[] = {1.1644f, 1.1644f, 1.1644f, 0.0f, -0.3917f,
                                     2.0172f, 1.5960f, -0.8129f, 0.0f};
    static const float bt601Full[] = {
        1.0f, 1.0f, 1.0f, 0.0f, -0.3441f, 1.7720f, 1.4020f, -0.7141f, 0.0f};
    static const float bt709Lim[] = {1.1644f, 1.1644f, 1.1644f, 0.0f, -0.2132f,
                                     2.1124f, 1.7927f, -0.5329f, 0.0f};
    static const float bt709Full[] = {
        1.0f, 1.0f, 1.0f, 0.0f, -0.1873f, 1.8556f, 1.5748f, -0.4681f, 0.0f};
    static const float bt2020Lim[] = {1.1644f, 1.1644f, 1.1644f,
                                      0.0f, -0.1874f, 2.1418f,
                                      1.6781f, -0.6505f, 0.0f};
    static const float bt2020Full[] = {
        1.0f, 1.0f, 1.0f, 0.0f, -0.1646f, 1.8814f, 1.4746f, -0.5714f, 0.0f};
    switch (color_space) {
    case AVCOL_SPC_SMPTE170M:
```

```
    case AVCOL_SPC_BT470BG:
        return color_full ? bt601Full : bt601Lim;
    case AVCOL_SPC_BT709:
        return color_full ? bt709Full : bt709Lim;
    case AVCOL_SPC_BT2020_NCL:
    case AVCOL_SPC_BT2020_CL:
        return color_full ? bt2020Full : bt2020Lim;
    default:
        return bt601Lim;
    }
}
EglVideoRenderer::~EglVideoRenderer() {
}
bool EglVideoRenderer::initialize(DECODER_PARAMETERS *params) {
    m_width = params->width;
    m_height = params->height;
    if (0 < m_width) {
        m_widthPercent = FIFTY_PERCENT * m_height / m_width;
    }
    if (params->context == nullptr) {
        eglLog(LOG_INFO, "EglContextInit execute");
        return false;
    }
    OHNativeWindow *window = static_cast<OHNativeWindow *>(params->context);
    if ((nullptr == window) || (0 >= params->width) || (0 >= params->height)) {
        eglLog(LOG_ERROR, "EglContextInit: param error");
        return false;
    }
    m_eglWindow = static_cast<EGLNativeWindowType>(window);
    if (nullptr == m_eglWindow) {
        eglLog(LOG_ERROR, "m_eglWindow is null");
        return false;
    }
    m_eglDisplay = eglGetDisplay(EGL_DEFAULT_DISPLAY);
    if (EGL_TRUE != eglInitialize(m_eglDisplay, 0, 0)) {
        eglLog(LOG_ERROR, "eglInitialize failed");
        return false;
    }
    EGLConfig eglConfig;
    EGLint configNum;
    EGLint configSpec[] = {
        EGL_RED_SIZE, 8,
        EGL_GREEN_SIZE, 8,
        EGL_BLUE_SIZE, 8,
        EGL_SURFACE_TYPE, EGL_WINDOW_BIT,
        EGL_NONE};
    if (EGL_TRUE != eglChooseConfig(m_eglDisplay, configSpec, &eglConfig, 1, &configNum)) {
        eglLog(LOG_ERROR, "eglChooseConfig failed");
        return false;
    }
    m_eglSurface = eglCreateWindowSurface(m_eglDisplay, eglConfig, m_eglWindow, nullptr);
    if (m_eglSurface == EGL_NO_SURFACE) {
        eglLog(LOG_ERROR, "eglCreateWindowSurface failed");
        return false;
    }
    const EGLint ctxAttr[] = {
        EGL_CONTEXT_CLIENT_VERSION, 2, EGL_NONE};
    m_eglContext = eglCreateContext(m_eglDisplay, eglConfig, EGL_NO_CONTEXT, ctxAttr);
    if (m_eglContext == EGL_NO_CONTEXT) {
        eglLog(LOG_ERROR, "eglCreateContext failed");
        return false;
    }
    if (EGL_TRUE != eglMakeCurrent(m_eglDisplay, m_eglSurface, m_eglSurface, m_eglContext)) {
        eglLog(LOG_ERROR, "eglMakeCurrent failed");
        return false;
    }
    Shader shader(vertexShaderWithMatrix, fragYUV420P);
    m_program = shader.use();
    if (PROGRAM_ERROR == m_program) {
        eglLog(LOG_ERROR, "CreateProgram: unable to create program");
        return false;
    }
    static float ver[] = {
        1.0f, -1.0f, 0.0f,
        -1.0f, -1.0f, 0.0f,
        1.0f, 1.0f, 0.0f,
        -1.0f, 1.0f, 0.0f};
    GLuint apos = static_cast<GLuint>(glGetAttribLocation(m_program, "aPosition"));
    glEnableVertexAttribArray(apos);
    glVertexAttribPointer(apos, 3, GL_FLOAT, GL_FALSE, 0, ver);
    static float fragment[] = {
        1.0f, 0.0f,
        0.0f, 0.0f,
        1.0f, 1.0f,
        0.0f, 1.0f};
    GLuint aTex = static_cast<GLuint>(glGetAttribLocation(m_program, "aTextCoord"));
    glEnableVertexAttribArray(aTex);
    glVertexAttribPointer(aTex, 2, GL_FLOAT, GL_FALSE, 0, fragment);
    int width = this->m_width;
    int height = this->m_height;
    glUniform1i(glGetUniformLocation(m_program, "yTexture"), 0);
    glUniform1i(glGetUniformLocation(m_program, "uTexture"), 1);
    glUniform1i(glGetUniformLocation(m_program, "vTexture"), 2);
    m_texture_id[3] = {0};
    glGenTextures(3, m_texture_id);
    glBindTexture(GL_TEXTURE_2D, m_texture_id[0]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexImage2D(GL_TEXTURE_2D,
                0,              // 细节基本 默认 0
                GL_LUMINANCE,   // gpu 内部格式 亮度，灰度图（这里就是只取一个亮度的颜色通道的意思）
                width,          // 加载的纹理宽度。最好为 2 的次幂(这里对 y 分量数据当做指定尺寸算，但显示尺寸会拉
伸到全屏？)
                height,         // 加载的纹理高度。最好为 2 的次幂
                0,              // 纹理边框
                GL_LUMINANCE,   // 数据的像素格式 亮度，灰度图
                GL_UNSIGNED_BYTE, // 像素点存储的数据类型
                NULL            // 纹理的数据（先不传）
    );
    glBindTexture(GL_TEXTURE_2D, m_texture_id[1]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexImage2D(GL_TEXTURE_2D,
                 0,          // 细节基本 默认 0
                 GL_LUMINANCE, // gpu 内部格式 亮度，灰度图（这里就是只取一个颜色通道的意思）
                 width / 2,   // u 数据数量为屏幕的 4 分之 1
                 height / 2,
                 0,            // 边框
                 GL_LUMINANCE,    // 数据的像素格式 亮度，灰度图
                 GL_UNSIGNED_BYTE, // 像素点存储的数据类型
                 NULL          // 纹理的数据（先不传）
    );
    glBindTexture(GL_TEXTURE_2D, m_texture_id[2]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexImage2D(GL_TEXTURE_2D,
                 0,          // 细节基本 默认 0
                 GL_LUMINANCE, // gpu 内部格式 亮度，灰度图（这里就是只取一个颜色通道的意思）
                 width / 2,
                 height / 2,     // v 数据数量为屏幕的 4 分之 1
                 0,            // 边框
                 GL_LUMINANCE,    // 数据的像素格式 亮度，灰度图
                 GL_UNSIGNED_BYTE, // 像素点存储的数据类型
                 NULL          // 纹理的数据（先不传）
    );
    return true;
}
void EglVideoRenderer::renderFrame(AVFrame *frame) {
    int width = m_width;
    int height = m_height;
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, m_texture_id[0]);
    glTexSubImage2D(GL_TEXTURE_2D, 0,
                    0, 0,        // 相对原来的纹理的 offset
                    width, height, // 加载的纹理宽度、高度。最好为 2 的次幂
                    GL_LUMINANCE, GL_UNSIGNED_BYTE,
                    frame->data[0]);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, m_texture_id[1]);
    glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, width / 2, height / 2, GL_LUMINANCE,
                    GL_UNSIGNED_BYTE,
                    frame->data[1]);
    glActiveTexture(GL_TEXTURE2);
    glBindTexture(GL_TEXTURE_2D, m_texture_id[2]);
    glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, width / 2, height / 2, GL_LUMINANCE,
                    GL_UNSIGNED_BYTE,
                    frame->data[2]);
    glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
    eglSwapBuffers(m_eglDisplay, m_eglSurface);
}
void EglVideoRenderer::bindTexture(int id) {
    float borderColorInternal[] = {borderColor[id], 0.0f, 0.0f, 1.0f};
    glBindTexture(GL_TEXTURE_2D, m_texture_id[id]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    textureWidth[id] = id > 0 ? m_frame_width / 2 : m_frame_width;
    textureHeight[id] = id > 0 ? m_frame_height / 2 : m_frame_height;
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RED, textureWidth[id], textureHeight[id],
                 0, GL_RED, GL_UNSIGNED_BYTE, nullptr);
    glUniform1i(m_texture_uniform[id], id);
}
void EglVideoRenderer::checkAndUpdateScale(AVFrame *frame) {
    if ((m_frame_width != frame->width) || (m_frame_height != frame->height)) {
        m_frame_width = frame->width;
        m_frame_height = frame->height;
        glBindBuffer(GL_ARRAY_BUFFER, m_vbo);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
                     GL_STATIC_DRAW);
        int positionLocation =
            glGetAttribLocation(m_program, "a_position");
        glEnableVertexAttribArray(positionLocation);
        glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, nullptr);
        for (int i = 0; i < 3; i++) {
            if (m_texture_id[i]) {
                glDeleteTextures(1, &m_texture_id[i]);
            }
        }
        glGenTextures(3, m_texture_id);
        for (int i = 0; i < 3; i++) {
            bindTexture(i);
        }
        bool colorFull = frame->color_range == AVCOL_RANGE_JPEG;
        glUniform3fv(m_offset_location, 1, gl_color_offset(colorFull));
        glUniformMatrix3fv(m_yuvmat_location, 1, GL_FALSE,
                           gl_color_matrix(frame->colorspace, colorFull));
        float frameAspect = ((float)m_frame_height / (float)m_frame_width);
        float screenAspect = ((float)m_height / (float)m_width);
        if (frameAspect > screenAspect) {
            float multiplier = frameAspect / screenAspect;
            glUniform4f(m_uv_data_location, 0.5f - 0.5f * (1.0f / multiplier),
                        0.0f, multiplier, 1.0f);
        } else {
            float multiplier = screenAspect / frameAspect;
            glUniform4f(m_uv_data_location, 0.0f,
                        0.5f - 0.5f * (1.0f / multiplier), 1.0f, multiplier);
        }
    }
}
```

AVFrameHolder.cpp

```
#include "video/AVFrameHolder.h"
AVFrameHolder::AVFrameHolder() {

}
AVFrameHolder* AVFrameHolder::m_holder = nullptr;
AVFrameHolder *AVFrameHolder::GetInstance() {
    if (m_holder == nullptr) {
        m_holder = new AVFrameHolder();
        return m_holder;
    } else {
        return m_holder;
    }
```

```
}
FFmpegVideoDecoder.cpp
#include <Limelight.h>
#include "FFmpegVideoDecoder.h"
#include <hilog/log.h>
#include <stdlib.h>
#include "video/AVFrameHolder.h"
#define ffDecodeLog(...)  OH_LOG_Print(LOG_APP, LOG_INFO, LOG_DOMAIN, "testTag", __VA_ARGS__);
#define DECODER_BUFFER_SIZE 92 * 1024 * 2
FFmpegVideoDecoder::FFmpegVideoDecoder() {}
FFmpegVideoDecoder::~FFmpegVideoDecoder() {}
DECODER_PARAMETERS* FFmpegVideoDecoder::getParams(){
    return &m_params;
}
int FFmpegVideoDecoder::setup(DECODER_PARAMETERS dparams) {
    m_params = dparams;
    DECODER_PARAMETERS* params = &dparams;
    m_stream_fps = params->frame_rate;
    ffDecodeLog(
        "FFmpeg: Setup with format: %{public}s, width: %{public}d, height: %{public}d, fps: %{public}d",
        params->video_format == VIDEO_FORMAT_H264 ? "H264" : "HEVC",
        params->width, params->height,
        params->frame_rate);
    av_log_set_level(AV_LOG_DEBUG);
#if LIBAVCODEC_VERSION_INT < AV_VERSION_INT(58, 10, 100)
    avcodec_register_all();
#endif
    m_packet = av_packet_alloc();
    int perf_lvl = LOW_LATENCY_DECODE;
    switch (params->video_format) {
    case VIDEO_FORMAT_H264:
        m_decoder = avcodec_find_decoder_by_name("h264");
        break;
    case VIDEO_FORMAT_H265:
        m_decoder = avcodec_find_decoder_by_name("hevc");
        break;
    }
    if (m_decoder == NULL) {
        ffDecodeLog("FFmpeg: Couldn't find decoder");
        return -1;
    }
    m_decoder_context = avcodec_alloc_context3(m_decoder);
    if (m_decoder_context == NULL) {
        ffDecodeLog("FFmpeg: Couldn't allocate context");
        return -1;
    }
    m_decoder_context->width = params->width;
    m_decoder_context->height = params->height;
    m_decoder_context->pix_fmt = AV_PIX_FMT_VIDEOTOOLBOX;
    int err = avcodec_open2(m_decoder_context, m_decoder, NULL);
    if (err < 0) {
        ffDecodeLog("FFmpeg: Couldn't open codec");
        return err;
    }
    m_frames_count = 2;
    m_frames = (AVFrame **)malloc(m_frames_count * sizeof(AVFrame *));
    if (m_frames == NULL) {
        ffDecodeLog("FFmpeg: Couldn't allocate frames");
        return -1;
    }
    tmp_frame = av_frame_alloc();
    for (int i = 0; i < m_frames_count; i++) {
        m_frames[i] = av_frame_alloc();
        if (m_frames[i] == NULL) {
            ffDecodeLog("FFmpeg: Couldn't allocate frame");
            return -1;
        }
        m_frames[i]->format = AV_PIX_FMT_YUV420P;
        m_frames[i]->width = params->width;
        m_frames[i]->height = params->height;
        int err = av_frame_get_buffer(m_frames[i], 256);
        if (err < 0) {
            ffDecodeLog("FFmpeg: Couldn't allocate frame buffer:");
            return -1;
        }
    }
    if (perf_lvl & DISABLE_LOOP_FILTER)
        m_decoder_context->skip_loop_filter = AVDISCARD_ALL;
    if (perf_lvl & LOW_LATENCY_DECODE)
        m_decoder_context->flags |= AV_CODEC_FLAG_LOW_DELAY;
    m_ffmpeg_buffer =
        (char *)malloc(DECODER_BUFFER_SIZE + AV_INPUT_BUFFER_PADDING_SIZE);
    if (m_ffmpeg_buffer == NULL) {
        ffDecodeLog("FFmpeg: Not enough memory");
        cleanup();
        return -1;
    }
    ffDecodeLog("FFmpeg: Setup done!");
    return DR_OK;
}
void FFmpegVideoDecoder::cleanup() {
    ffDecodeLog("FFmpeg: Cleanup...");
    av_packet_free(&m_packet);
    if (hw_device_ctx) {
        av_buffer_unref(&hw_device_ctx);
    }
    if (m_decoder_context) {
        avcodec_close(m_decoder_context);
        av_free(m_decoder_context);
        m_decoder_context = NULL;
    }
    if (m_frames) {
        for (int i = 0; i < m_frames_count; i++) {
            if (m_frames[i])
                av_frame_free(&m_frames[i]);
        }
        free(m_frames);
        m_frames = nullptr;
    }
    if (tmp_frame) {
        av_frame_free(&tmp_frame);
    }
    if (m_ffmpeg_buffer) {
        free(m_ffmpeg_buffer);
        m_ffmpeg_buffer = nullptr;
    }
    ffDecodeLog("FFmpeg: Cleanup done!");
}
int FFmpegVideoDecoder::submitDecodeUnit(PDECODE_UNIT du) {
    if (m_frames_in == 0 && du->frameType != FRAME_TYPE_IDR) {
        return DR_NEED_IDR;
    }
    if (du->fullLength < DECODER_BUFFER_SIZE) {
        PLENTRY entry = du->bufferList;
        if (!m_last_frame) {
            m_video_decode_stats.measurementStartTimestamp = LiGetMillis();
            m_last_frame = du->frameNumber;
        } else {
            m_video_decode_stats.networkDroppedFrames +=
                du->frameNumber - (m_last_frame + 1);
            m_video_decode_stats.totalFrames +=
                du->frameNumber - (m_last_frame + 1);
            m_last_frame = du->frameNumber;
        }
        m_video_decode_stats.receivedFrames++;
        m_video_decode_stats.totalFrames++;
        int length = 0;
        while (entry != NULL) {
            if (length > DECODER_BUFFER_SIZE) {
            }
            memcpy(m_ffmpeg_buffer + length, entry->data, entry->length);
            length += entry->length;
            entry = entry->next;
        }
        m_video_decode_stats.totalReassemblyTime +=
            LiGetMillis() - du->receiveTimeMs;
        m_frames_in++;
        uint64_t before_decode = LiGetMillis();
        if (length > DECODER_BUFFER_SIZE) {
            ffDecodeLog("FFmpeg: Big buffer to decode...");
        }
        if (du->frameType == FRAME_TYPE_IDR) {
            m_packet->flags = AV_PKT_FLAG_KEY;
        } else {
            m_packet->flags = 0;
        }
        if (decode(m_ffmpeg_buffer, length) == 0) {
            m_frames_out++;
            m_video_decode_stats.totalDecodeTime +=
                LiGetMillis() - before_decode;
            m_video_decode_stats.totalDecodeTime +=
                (m_frames_in - m_frames_out) * (1000 / m_stream_fps);
            m_video_decode_stats.decodedFrames++;
            m_frame = get_frame(true);
            ffDecodeLog("frame size %{public}d X %{public}d", m_frame->width, m_frame->height);
            AVFrameHolder::GetInstance()->push(m_frame);
        }
    } else {
        ffDecodeLog("FFmpeg: Big buffer to decode... 2");
    }
    return DR_OK;
}
int FFmpegVideoDecoder::decode(char *indata, int inlen) {
    m_packet->data = (uint8_t *)indata;
    m_packet->size = inlen;
    int err = avcodec_send_packet(m_decoder_context, m_packet);
    if (err != 0) {
        char error[512];
        av_strerror(err, error, sizeof(error));
        char *message = error;
        ffDecodeLog("FFmpeg: Decode failed - %{public}s", message);
    }
    return err != DR_OK ? err : DR_OK;
}
AVFrame *FFmpegVideoDecoder::get_frame(bool native_frame) {
    int err = avcodec_receive_frame(m_decoder_context, tmp_frame);
    if (hw_device_ctx) {
        if ((err = av_hwframe_transfer_data(m_frames[m_next_frame], tmp_frame, 0)) < 0) {
            ffDecodeLog("FFmpeg: Error transferring the data to system memory with error {}", err);
            return NULL;
        }
        av_frame_copy_props(m_frames[m_next_frame], tmp_frame);
    } else {
        m_frames[m_next_frame] = tmp_frame;
    }
    if (err == 0) {
        m_current_frame = m_next_frame;
        m_next_frame = (m_current_frame + 1) % m_frames_count;
        if (/*ffmpeg_decoder == SOFTWARE ||*/ native_frame)
            return m_frames[m_current_frame];
    } else if (err != AVERROR(EAGAIN)) {
        char error[512];
        av_strerror(err, error, sizeof(error));
        ffDecodeLog("FFmpeg: Receive failed - %d/%s", err, error);
    }
    return NULL;
}
VIDEO_STATS *FFmpegVideoDecoder::video_decode_stats() {
    uint64_t now = LiGetMillis();
    m_video_decode_stats.totalFps =
        (float)m_video_decode_stats.totalFrames /
        ((float)(now - m_video_decode_stats.measurementStartTimestamp) /
         1000);
    m_video_decode_stats.receivedFps =
        (float)m_video_decode_stats.receivedFrames /
        ((float)(now - m_video_decode_stats.measurementStartTimestamp) /
         1000);
    m_video_decode_stats.decodedFps =
        (float)m_video_decode_stats.decodedFrames /
        ((float)(now - m_video_decode_stats.measurementStartTimestamp) /
         1000);
    return (VIDEO_STATS *)&m_video_decode_stats;
}
```

x509Utils.cpp

```
#include "x509Utils.h"
#include <openssl/bio.h>
#include <openssl/err.h>
#include <openssl/pem.h>
#include <openssl/x509.h>
#include <openssl/x509v3.h>
#include <moon_bridge.h>
void THROW_BAD_ALLOC_IF_NULL(void *target) {
    if (target == nullptr) {
        ERR_print_errors_fp(stderr);
        abort();
    }
}
long getFileSize(FILE *file) {
    long size;
    long currentPosition = ftell(file);
    fseek(file, 0, SEEK_END);
    size = ftell(file);
    fseek(file, currentPosition, SEEK_SET);
    return size;
}
EVP_PKEY *generateKey() {
    EVP_PKEY_CTX *ctx = EVP_PKEY_CTX_new_id(EVP_PKEY_RSA, NULL);
    THROW_BAD_ALLOC_IF_NULL(ctx);
    EVP_PKEY_keygen_init(ctx);
    EVP_PKEY_CTX_set_rsa_keygen_bits(ctx, 2048);
    EVP_PKEY *pk = NULL;
    EVP_PKEY_keygen(ctx, &pk);
    EVP_PKEY_CTX_free(ctx);
    THROW_BAD_ALLOC_IF_NULL(pk);
    return pk;
}
int generate_x509_certificate(char *cert_path, char *key_path) {
    EVP_PKEY *pk = nullptr;
    X509 *cert = nullptr;
    FILE *cert_file = nullptr;
    FILE *key_file = nullptr;
    OpenSSL_add_all_algorithms();
    cert = X509_new();
    pk = generateKey();
    X509_set_version(cert, 2);
    ASN1_INTEGER_set(X509_get_serialNumber(cert), 0);
#if OPENSSL_VERSION_NUMBER < 0x10100000L
    X509_gmtime_adj(X509_get_notBefore(cert), 0);
    X509_gmtime_adj(X509_get_notAfter(cert), 60 * 60 * 24 * 365 * 20); // 20 yrs
#else
    ASN1_TIME *before = ASN1_STRING_dup(X509_get0_notBefore(cert));
    THROW_BAD_ALLOC_IF_NULL(before);
    ASN1_TIME *after = ASN1_STRING_dup(X509_get0_notAfter(cert));
    THROW_BAD_ALLOC_IF_NULL(after);
    X509_gmtime_adj(before, 0);
    X509_gmtime_adj(after, 60 * 60 * 24 * 365 * 20); // 20 yrs
    X509_set1_notBefore(cert, before);
    X509_set1_notAfter(cert, after);
    ASN1_STRING_free(before);
    ASN1_STRING_free(after);
```

```
#endif
    X509_set_pubkey(cert, pk);
    X509_NAME *name = X509_get_subject_name(cert);
    X509_NAME_add_entry_by_txt(name, "CN", MBSTRING_ASC,
                               reinterpret_cast<unsigned char *>(const_cast<char *>("NVIDIA GameStream Client")),
                               -1, -1, 0);
    X509_set_issuer_name(cert, name);
    X509_sign(cert, pk, EVP_sha256());
    cert_file = fopen(cert_path, "w");
    int ret = PEM_write_X509(cert_file, cert);
    fclose(cert_file);
    key_file = fopen(key_path, "w");
    ret = PEM_write_PrivateKey(key_file, pk, nullptr, nullptr, 0, nullptr, nullptr);
    fclose(key_file);
    FILE *key_cer_file = nullptr;
    key_cer_file = fopen(strcat(key_path, ".cer"), "w");
    ret = i2d_PrivateKey_fp(key_cer_file, pk);
    fclose(key_cer_file);
    EVP_PKEY_free(pk);
    X509_free(cert);
    EVP_cleanup();
    return 0;
}
napi_value generate_certificate(napi_env env, napi_callback_info info) {
    size_t argc = 2;
    napi_value args[2] = {nullptr};
    napi_get_cb_info(env, info, &argc, args, nullptr, nullptr);
    char *certPath = get_value_string(env, args[0]);
    char *keyPath = get_value_string(env, args[1]);
    generate_x509_certificate(certPath, keyPath);
    return 0;
}
napi_value verifySignature(napi_env env, napi_callback_info info) {
    size_t argc = 3;
    napi_value args[3] = {nullptr};
    napi_get_cb_info(env, info, &argc, args, nullptr, nullptr);
    void *data;
    void *signature;
    void *serverCertificate;
    size_t dataLength;
    size_t signatureLength;
    size_t serverCertificateLength;
    napi_get_typedarray_info(
        env,
        args[0],
        nullptr,
        &dataLength,
        &data,
        nullptr, // 可选的 ArrayBuffer
        nullptr  // 可选的偏移
    );
    napi_get_typedarray_info(
        env,
        args[1],
        nullptr,
        &signatureLength,
        &signature,
        nullptr, // 可选的 ArrayBuffer
        nullptr  // 可选的偏移
    );
    napi_get_typedarray_info(
        env,
        args[2],
        nullptr,
        &serverCertificateLength,
        &serverCertificate,
        nullptr, // 可选的 ArrayBuffer
        nullptr  // 可选的偏移
    );
    BIO *bio = BIO_new_mem_buf(serverCertificate, serverCertificateLength);
    THROW_BAD_ALLOC_IF_NULL(bio);
    X509 *cert = PEM_read_bio_X509(bio, nullptr, nullptr, nullptr);
    BIO_free_all(bio);
    EVP_PKEY *pubKey = X509_get_pubkey(cert);
    THROW_BAD_ALLOC_IF_NULL(pubKey);
    EVP_MD_CTX *mdctx = EVP_MD_CTX_create();
    THROW_BAD_ALLOC_IF_NULL(mdctx);
    EVP_DigestVerifyInit(mdctx, nullptr, EVP_sha256(), nullptr, pubKey);
    EVP_DigestVerifyUpdate(mdctx, data, dataLength);
    int result = EVP_DigestVerifyFinal(mdctx, reinterpret_cast<unsigned char *>(signature), signatureLength);
    EVP_PKEY_free(pubKey);
    EVP_MD_CTX_destroy(mdctx);
    X509_free(cert);
    napi_value ret;
    napi_get_boolean(env, result > 0, &ret);
    return ret;
}
napi_value createTypedArray(napi_env env, size_t length, napi_typedarray_type type, void *data) {
    napi_value arrayBuffer;
    void *arrayBufferPtr;
    napi_value uint8Array;
    napi_status status = napi_create_arraybuffer(env, length, &arrayBufferPtr, &arrayBuffer);
    if (status != napi_ok) {
        return NULL;
    }
    memcpy(arrayBufferPtr, data, length);
    status = napi_create_typedarray(env, type, length, arrayBuffer, 0, &uint8Array);
    if (status != napi_ok) {
        return NULL;
    }
    return uint8Array;
}
napi_value signMessage(napi_env env, napi_callback_info info) {
    size_t argc = 2;
    napi_value args[2] = {nullptr};
    napi_get_cb_info(env, info, &argc, args, nullptr, nullptr);
    void *message;
    size_t messageLength;
    void *privateKey;
    size_t privateKeyLength;
    napi_get_typedarray_info(
        env,
        args[0],
        nullptr,
        &messageLength,
        &message,
        nullptr, // 可选的 ArrayBuffer
        nullptr  // 可选的偏移
    );
    napi_get_typedarray_info(
        env,
        args[1],
        nullptr,
        &privateKeyLength,
        &privateKey,
        nullptr,
        nullptr);
    EVP_MD_CTX *ctx = EVP_MD_CTX_create();
    THROW_BAD_ALLOC_IF_NULL(ctx);
    BIO *bio = BIO_new_mem_buf(privateKey, privateKeyLength);
    THROW_BAD_ALLOC_IF_NULL(bio);
    EVP_PKEY *m_PrivateKey = PEM_read_bio_PrivateKey(bio, nullptr, nullptr, nullptr);
    BIO_free_all(bio);
    EVP_DigestSignInit(ctx, NULL, EVP_sha256(), NULL, m_PrivateKey);
    EVP_DigestSignUpdate(ctx, reinterpret_cast<unsigned char *>(message), messageLength);
    size_t signatureLength = 0;
    EVP_DigestSignFinal(ctx, NULL, &signatureLength);
    unsigned char *signature = (unsigned char *)malloc(signatureLength);
    EVP_DigestSignFinal(ctx, signature, &signatureLength);
    napi_value uint8Array =
        createTypedArray(env, signatureLength, napi_uint8_array, signature);
    EVP_MD_CTX_destroy(ctx);
    return uint8Array;
}
napi_value getSignatureFromPemCert(napi_env env, napi_callback_info info) {
    size_t argc = 1;
    napi_value args[1] = {nullptr};
    napi_get_cb_info(env, info, &argc, args, nullptr, nullptr);
    void *serverCertificate;
    size_t serverCertificateLength;
    napi_get_typedarray_info(
        env,
        args[0],
        nullptr,
        &serverCertificateLength,
        &serverCertificate,
        nullptr, // 可选的 ArrayBuffer
        nullptr  // 可选的偏移
    );
    BIO *bio = BIO_new_mem_buf(serverCertificate, serverCertificateLength);
    THROW_BAD_ALLOC_IF_NULL(bio);
    X509 *cert = PEM_read_bio_X509(bio, nullptr, nullptr, nullptr);
    BIO_free_all(bio);
#if (OPENSSL_VERSION_NUMBER < 0x10002000L)
    ASN1_BIT_STRING *asnSignature = cert->signature;
#elif (OPENSSL_VERSION_NUMBER < 0x10100000L)
    ASN1_BIT_STRING *asnSignature;
    X509_get0_signature(&asnSignature, NULL, cert);
```

```
#else
    const ASN1_BIT_STRING *asnSignature;
    X509_get0_signature(&asnSignature, NULL, cert);
#endif
    X509_free(cert);
    return createTypedArray(env, asnSignature->length, napi_uint8_array, asnSignature->data);
}
napi_value encrypt(napi_env env, napi_callback_info info) {
    size_t argc = 2;
    napi_value args[2] = {nullptr};
    napi_get_cb_info(env, info, &argc, args, nullptr, nullptr);
    void *message;
    size_t messageLength;
    void *privateKey;
    size_t privateKeyLength;
    napi_get_typedarray_info(
        env,
        args[0],
        nullptr,
        &messageLength,
        &message,
        nullptr, // 可选的 ArrayBuffer
        nullptr  // 可选的偏移
    );
    napi_get_typedarray_info(
        env,
        args[1],
        nullptr,
        &privateKeyLength,
        &privateKey,
        nullptr,
        nullptr);
    void *ciphertext = malloc(messageLength);
    EVP_CIPHER_CTX *cipher;
    int ciphertextLen;
    cipher = EVP_CIPHER_CTX_new();
    THROW_BAD_ALLOC_IF_NULL(cipher);
    EVP_EncryptInit(cipher, EVP_aes_128_ecb(), reinterpret_cast<const unsigned char *>(privateKey), NULL);
    EVP_CIPHER_CTX_set_padding(cipher, 0);
    EVP_EncryptUpdate(cipher,
                      reinterpret_cast<unsigned char *>(ciphertext),
                      &ciphertextLen,
                      reinterpret_cast<const unsigned char *>(message),
                      messageLength);
    EVP_CIPHER_CTX_free(cipher);
    return createTypedArray(env, messageLength, napi_uint8_array, ciphertext);
}
napi_value decrypt(napi_env env, napi_callback_info info) {
    size_t argc = 2;
    napi_value args[2] = {nullptr};
    napi_get_cb_info(env, info, &argc, args, nullptr, nullptr);
    void *message;
    size_t messageLength;
    void *privateKey;
    size_t privateKeyLength;
    napi_get_typedarray_info(
        env,
        args[0],
        nullptr,
        &messageLength,
        &message,
        nullptr, // 可选的 ArrayBuffer
        nullptr  // 可选的偏移
    );
    napi_get_typedarray_info(
        env,
        args[1],
        nullptr,
        &privateKeyLength,
        &privateKey,
        nullptr,
        nullptr);
    void *plaintext = malloc(messageLength);
    EVP_CIPHER_CTX *cipher;
    int plaintextLen;
    cipher = EVP_CIPHER_CTX_new();
    THROW_BAD_ALLOC_IF_NULL(cipher);
    EVP_DecryptInit(cipher, EVP_aes_128_ecb(), reinterpret_cast<const unsigned char *>(privateKey), NULL);
    EVP_CIPHER_CTX_set_padding(cipher, 0);
    EVP_DecryptUpdate(cipher,
                      reinterpret_cast<unsigned char *>(plaintext),
                      &plaintextLen,
                      reinterpret_cast<const unsigned char *>(message),
                      messageLength);
    EVP_CIPHER_CTX_free(cipher);
    return createTypedArray(env, messageLength, napi_uint8_array, plaintext);
}
```

http_curl.cpp

```
#include "http_curl.h"
#include "moonlight-core/moon_bridge.h"
#include <curl/easy.h>
#include <stdlib.h>
#include "string.h"
#include "napi/native_api.h"
#include "x509Utils.h"
struct AsyncCallbackInfo {
    napi_env env;
    napi_async_work asyncWork;
    napi_deferred deferred;
    const char *url;
    const int timeout;
    const char *clientPath;
    const char *keyPath;
    void *result;
    size_t size;
    const char *error;
};
struct HTTP_DATA {
    char *memory;
    size_t size;
};
size_t write_callback(void *contents, size_t size, size_t nmemb, void *userp) {
    size_t realsize = size * nmemb;
    HTTP_DATA *mem = (HTTP_DATA *)userp;
    mem->memory = (char *)realloc(mem->memory, mem->size + realsize + 1);
    if (mem->memory == NULL)
        return 0;
    memcpy(&(mem->memory[mem->size]), contents, realsize);
    mem->size += realsize;
    mem->memory[mem->size] = 0;
    return realsize;
}
static CURL *curl;
int http_init( AsyncCallbackInfo *cb) {
    if (!curl) {
        curl_global_init(CURL_GLOBAL_ALL);
    } else {
        return 0;
    }
    curl = curl_easy_init();
    if (!curl)
        return 1;
    curl_easy_setopt(curl, CURLOPT_SSL_VERIFYHOST, 0L);
    curl_easy_setopt(curl, CURLOPT_SSLENGINE_DEFAULT, 1L);
    curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 0L);
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_callback);
    curl_easy_setopt(curl, CURLOPT_FAILONERROR, 1L);
    curl_easy_setopt(curl, CURLOPT_SSL_SESSIONID_CACHE, 0L);
    return 0;
}
void http_request(AsyncCallbackInfo *cb) {
    HTTP_DATA *http_data = (HTTP_DATA *)malloc(sizeof(HTTP_DATA));
    http_data->memory = (char *)malloc(1);
    http_data->size = 0;
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, http_data);
    curl_easy_setopt(curl, CURLOPT_URL, cb->url);
    curl_easy_setopt(curl, CURLOPT_TIMEOUT, cb->timeout);
    if (cb->clientPath != nullptr) {
        curl_easy_setopt(curl, CURLOPT_SSLCERTTYPE, "PEM");
        curl_easy_setopt(curl, CURLOPT_SSLCERT, cb->clientPath);
    }
    if (cb->keyPath != nullptr) {
        curl_easy_setopt(curl, CURLOPT_SSLKEYTYPE, "PEM");
        curl_easy_setopt(curl, CURLOPT_SSLKEY, cb->keyPath);
    }
    CURLcode res = curl_easy_perform(curl);
    if (res != CURLE_OK) {
        cb->error = curl_easy_strerror(res);
    } else if (http_data->memory == NULL) {
        cb->error = "Curl: memory = NULL";
    }
    cb->result = http_data->memory;
    cb->size = http_data->size;
    free(http_data->memory);
    free(http_data);
}
void getCurl(napi_env env, AsyncCallbackInfo *cb) {
    CURL *curl;
    CURLcode res;
    curl = curl_easy_init();
    if (curl) {
        curl_easy_setopt(curl, CURLOPT_URL, cb->url);
        curl_easy_setopt(curl, CURLOPT_TIMEOUT, cb->timeout);
        curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);
        curl_easy_setopt(curl, CURLOPT_SSL_VERIFYPEER, 0L);
        curl_easy_setopt(curl, CURLOPT_SSL_VERIFYHOST, 0L);
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_callback);
        HTTP_DATA *http_data = (HTTP_DATA *)malloc(sizeof(HTTP_DATA));
        http_data->memory = (char *)malloc(1);
        http_data->size = 0;
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, http_data);
        res = curl_easy_perform(curl);
        if (res != CURLE_OK) {
            cb->error = curl_easy_strerror(res);
        } else if (http_data->memory == NULL) {
            cb->error = "Curl: memory = NULL";
        }
        cb->result = http_data->memory;
        cb->size = http_data->size;
        free(http_data->memory);
        free(http_data);
    }
}
napi_value GetRequest(napi_env env, napi_callback_info info) {
    napi_deferred deferred;
    napi_value promise;
    napi_create_promise(env, &deferred, &promise);
    size_t argc = 4;
    napi_value args[4] = {nullptr};
    napi_get_cb_info(env, info, &argc, args, nullptr, nullptr);
    char *url = get_value_string(env, args[0]);
    int timeout;
    napi_get_value_int32(env, args[1], &timeout);
    char *clientPath = get_value_string(env, args[2]);
    char *keyPath = get_value_string(env, args[3]);
    AsyncCallbackInfo *asyncCallbackInfo = new AsyncCallbackInfo{
        .env = env,
        .asyncWork = nullptr,
        .deferred = deferred,
        .url = url,
        .timeout = timeout,
        .clientPath = clientPath,
        .keyPath = keyPath,
        .result = nullptr};
    napi_value resourceName;
    napi_create_string_latin1(env, url, NAPI_AUTO_LENGTH, &resourceName);
    napi_create_async_work(
        env, nullptr, resourceName,
        [](napi_env env, void *data) {
            http_request((AsyncCallbackInfo *)data);
        },
        [](napi_env env, napi_status status, void *data) {
            AsyncCallbackInfo *asyncCallbackInfo = (AsyncCallbackInfo *)data;
            if (asyncCallbackInfo->error == nullptr) {
                napi_value result = createTypedArray(env, asyncCallbackInfo->size, napi_uint8_array,
asyncCallbackInfo->result);
                napi_resolve_deferred(asyncCallbackInfo->env, asyncCallbackInfo->deferred, result);
            } else {
                napi_value result;
                napi_create_string_utf8(env, asyncCallbackInfo->error, NAPI_AUTO_LENGTH, &result);
                napi_reject_deferred(env, asyncCallbackInfo->deferred, result);
            }
            napi_delete_async_work(env, asyncCallbackInfo->asyncWork);
            delete asyncCallbackInfo;
        },
        (void *)asyncCallbackInfo, &asyncCallbackInfo->asyncWork);
    napi_queue_async_work(env, asyncCallbackInfo->asyncWork);
    return promise;
}
static napi_value CurlClientClassConstructor(napi_env env, napi_callback_info info) {
    http_init(nullptr);
    napi_value thisArg = nullptr;
    void *data = nullptr;
    napi_get_cb_info(env, info, nullptr, nullptr, &thisArg, &data);
    napi_value global = nullptr;
    napi_get_global(env, &global);
    return thisArg;
}
static napi_value Close(napi_env env, napi_callback_info info) {

    return 0;
}
void HttpCurlInit(napi_env env, napi_value exports) {
    napi_property_descriptor descriptors[] = {
        {"get", nullptr, GetRequest, nullptr, nullptr, nullptr, napi_default, nullptr},
        {"close", nullptr, Close, nullptr, nullptr, nullptr, napi_default, nullptr}};
    napi_value result = nullptr;
    napi_define_class(env, "CurlClient", NAPI_AUTO_LENGTH, CurlClientClassConstructor, nullptr,
                      sizeof(descriptors) / sizeof(*descriptors), descriptors, &result);
    napi_set_named_property(env, exports, "CurlClient", result);
}
```

GamePage.ets

```
import gameViewModel from 'ets/entryability/GameViewModel'
import { VideoStatus } from 'libentry.so';
import router from '@ohos.router';
import { Spinner } from './compoments/Loading'
import { StreamSettings } from '../uitls/StreamSetttings';
import { loadSettings} from '../uitls/StreamSetttings';
import { VirtualController,  } from '../virtual_controller/VirtualController';
import { VirtualControllerConfigurationLoader } from '../virtual_controller/VirtualControllerConfigurationLoader';
import { VirtualControllerButton } from '../virtual_controller/common';
import { Icon } from './compoments/Title';
@Entry
@Component
struct GamePage {
  @State videoStatus: VideoStatus = null
  xComponentContext: any | undefined = undefined;
  @State virtualController: VirtualController = null
  @State settings: StreamSettings = null
  dialogController: CustomDialogController =  new CustomDialogController({
```

```
  builder: Spinner({ title: $r('app.string.conn_establishing_title'), text: "" }),
  autoCancel: false,
  alignment: DialogAlignment.Center,
  customStyle: true
 })
 aboutToDisappear() {
  this.dialogController.close()
  gameViewModel.stop()
  delete this.dialogController, // 删除  dialogController
  this.dialogController = undefined // 将  dialogController 置空
 }
 async onInit(){
  this.dialogController.open()
  const settings = await loadSettings(getContext(this))
  this.settings = settings
  await gameViewModel.init(router.getParams()["computer"], router.getParams()["app"], settings)
  await gameViewModel.start(this.dialogController, getContext(this))
  if (settings.touchscreen_trackpad){
   const loader = new VirtualControllerConfigurationLoader()
   loader.createDefaultLayout(gameViewModel.virtualController, settings)
   VirtualControllerConfigurationLoader.loadFromPreferences(gameViewModel.virtualController, getContext(this))
   this.virtualController = gameViewModel.virtualController
  }
  gameViewModel.conn.onVideoStatus((s)=>{
   this.videoStatus = s;
  })
 }
 aboutToAppear(){
  this.onInit()
 }
 build() {
  Stack({ alignContent: Alignment.TopStart }){
   XComponent({ id: 'xcomponentId1', type: 'surface', libraryname: 'entry' })
    .onLoad((context: any) => {
     this.xComponentContext = context
    })
    .onDestroy(() => {
     console.log("onDestroy");
    }).borderWidth(0).height("100%").width('100%')
   if(this.videoStatus){
    Column(){
     Text(`视频流:  ${this.settings?.resolution_list} ${this.videoStatus?.totalFps?.toFixed(2) || 0 }
FPS`).fontColor(Color.White)
     Text("解码器: ").fontColor(Color.White)
     Text(`网络接收帧数: ${this.videoStatus?.receivedFps?.toFixed(2) || 0 } FPS`) .fontColor(Color.White)
     Text(`渲染帧数: ${this.videoStatus?.renderedFps?.toFixed(2) || 0 } FPS`).fontColor(Color.White)
     Text(`网络丢失帧:  ${this.videoStatus?.networkDroppedRate?.toFixed(2) || 0 } %`).fontColor(Color.White)
     Text(`平均网络延迟: ${this.videoStatus?.receivedTime?.toFixed(2) || 0 } `).fontColor(Color.White)
     Text(`平均解码时间:  ${this.videoStatus?.decodeTime?.toFixed(2) || 0 }  ms`).fontColor(Color.White)
    }.backgroundColor(Color.Black).opacity(0.5)
   }
  if(this.virtualController){
   ForEach(this.virtualController.elements.convertToArray(), (d) => {
    VirtualControllerButton({ element: d, layout: d.layout })
   }, (d) =>d.elementId.toString())
   Button(){
        Icon({icon:$r("app.media.settings")})
      }.offset({y: 20}).onClick(()=>{
        this.virtualController.onSettingsClick(getContext(this))
      })
     }
    }.backgroundColor(Color.Black)
    .width('100%')
    .height('100%')
  }
}
```

AppPage.ets

```
import router from '@ohos.router'
import viewModel from '../entryability/ComputerManagerViewModel'
import { ComputerDetails } from '../entryability/computers/ComputerDetails'
import { NvHttp } from '../entryability/http/NvHttp'
import { Icon} from './compoments/Title'
import { NavTitle } from './compoments/Title'
import { NvApp } from '../entryability/http/NvApp'
import image from '@ohos.multimedia.image'
@Component
struct AppView {
  computer: ComputerDetails
  app: NvApp
  @State image: PixelMap = undefined;
  @Builder
  pcMenu() {
    Menu() {
      MenuItem({ content: $r("app.string.applist_menu_resume") })
      MenuItem({ content: $r("app.string.applist_menu_quit")  })
      MenuItem({ content: $r("app.string.applist_menu_quit_and_start") })
    }
  }
  aboutToAppear(){
    viewModel.readImageByDisk(this.app).then((res)=>{
      let options = {
        alphaType: 0,                    // 透明度
        editable: false,                 // 是否可编辑
        pixelFormat: 3,                  // 像素格式
        scaleMode: 1,                    // 缩略值
        size: { height: 100, width: 100}
      }
      let imageSource = image.createImageSource(res.buffer);
      if(imageSource)
        imageSource.createPixelMap(options).then((pixelMap) => {
          this.image = pixelMap
        })
    })
  }
  build(){
    Column(){
      Stack(){
        if (this.image){
          Image(this.image).height(150).width(100)
        } else {
          Text(this.app.appName).fontColor(Color.White)
        }
        if(this.computer.runningGameId == this.app.appId){
          Column(){
            Icon({icon: $r("app.media.play_arrow_FILL1_wght700_GRAD200_opsz48"), iconSize:48})
            Icon({icon: $r("app.media.stop_FILL1_wght700_GRAD200_opsz48"), iconSize: 48})
          }
        }
      }
    }.onClick(()=>{
        router.pushUrl({ url:"pages/GamePage", params: { app: this.app, computer: this.computer } })
    })
  }
}
@Entry
@Component
struct AppPage {
  computer: ComputerDetails
  @State appList: NvApp[] = []
  aboutToAppear(){
    const params = router.getParams();
    viewModel.getComputerByUUid(params["uuid"]).then((d)=>{
      this.computer = d
      const appList = d.appList
      this.appList = appList.filter((d) => d.appId != null);
    })
  }
  build(){
    Column(){
      NavTitle({ title: router.getParams()["computerName"] }).width("100%")
      Grid() {
        ForEach(this.appList, (d:NvApp)=>{
          GridItem(){
            AppView({app: d, computer: this.computer})
          }
        },(item) => JSON.stringify(item))
      }
      .rowsTemplate('1fr 1fr 1fr')
      .columnsTemplate('1fr 1fr 1fr')
    }.padding(10).height("100%").width("100%").backgroundColor($r("app.color.page_background"))
  }
}
AddPage.ets
import { NavTitle } from './compoments/Title'
import viewModel from '../entryability/ComputerManagerViewModel'
import router from '@ohos.router'
import { Alert } from './compoments/Loading'
import { Loading} from './compoments/Loading'
import promptAction from '@ohos.promptAction'
@Entry
@Component
struct AddPage {
  viewModel = viewModel
  @State ip: string = ''
  loadingDialog: CustomDialogController = new CustomDialogController({
    builder: Loading({
      title: $r("app.string.title_add_pc"),
      text: $r("app.string.msg_add_pc"),
    }),
    autoCancel: false,
    alignment: DialogAlignment.Center,
    customStyle: true
  })
  build(){
    Column(){
      NavTitle({ title: $r("app.string.title_add_pc")})
      Row(){
        TextInput({placeholder:'串流电脑的 ip 地址', text:
"[2409:8a00:79a1:5ae1:27d5:8223:fcba:4b17]"}).fontColor(Color.White).onChange((value: string) => {
          this.ip = value;
        }).layoutWeight(1).type(InputType.Normal)
        Button("确定").width(150).margin({left:5}).onClick(()=>{
            this.loadingDialog.open()
            this.viewModel.addPc(this.ip).then((result)=>{
                this.loadingDialog.close()
                if(result.success){
                  router.back()
                }else{
                  promptAction.showDialog({ title:  $r('app.string.conn_error_title'), message: result.message})
                }
            }).catch(()=>{
                promptAction.showDialog({ title:  $r('app.string.conn_error_title'), message: "未知错误"})
                this.loadingDialog.close()
            })
        })
      }.width("100%")
    }.padding(20).height("100%").width("100%").backgroundColor($r("app.color.page_background"))
  }
}
SettingsPage.ets
import { NavTitle } from './compoments/Title';
import dataPreferences from '@ohos.data.preferences';
import { getResString } from '../uitls/ResString';
import { loadSettings } from '../uitls/StreamSetttings';
import { StreamSettings} from '../uitls/StreamSetttings';
import router from '@ohos.router';
@Entry
@Component
struct SettingsPage {
  scroller: Scroller = new Scroller();
  preferences: dataPreferences.Preferences = null
  @Provide settings: StreamSettings = null
  onUpdateValue = (key: string, value: any)=>{
    this.preferences.put(key, value)
    this.preferences.flush()
  }
  aboutToAppear(){
    this.loadPreferences(getContext(this))
  }
  async loadPreferences(context: Context){
    this.preferences = await dataPreferences.getPreferences(context, "StreamSettings")
    this.settings = await loadSettings(context)
  }
  build() {
```

```
  Column() {
    NavTitle({ title: "设置" })
    Scroll(this.scroller) {
      Column({ space: 10 }) {
        Column({ space: 10 }) {
          Text($r("app.string.category_basic_settings"))
          ListPreference({
            pKey: "resolution_list",
            title: $r("app.string.title_resolution_list"),
            summary: $r("app.string.summary_resolution_list"),
            names:[$r("app.string.resolution_360p"), $r("app.string.resolution_720p"),
$r("app.string.resolution_1080p"), $r("app.string.resolution_1440p"), $r("app.string.resolution_4k")],
            entries: ["640x360", "1280x720", "1920x1080", "2560x1440", "3840x2160"],
            value: "1280x720",
            onChange: this.onUpdateValue
          })
          ListPreference({
            pKey: "fps_list",
            title: $r("app.string.title_fps_list"),
            summary: $r("app.string.summary_fps_list"),
            value: "60",
            names:[$r("app.string.fps_30"), $r("app.string.fps_60"), $r("app.string.fps_90"), $r("app.string.fps_120")],
            entries: ["30", "60", "90", "120"],
            onChange: this.onUpdateValue
          })
          SeekBarPreference({
            pKey: "seekbar_bitrate",
            title: $r("app.string.title_seekbar_bitrate"),
            summary: $r("app.string.summary_seekbar_bitrate"),
            value: "",
            onChange: this.onUpdateValue
          })
          ListPreference({
            pKey: "frame_pacing",
            title: $r("app.string.title_frame_pacing"),
            summary: $r("app.string.summary_frame_pacing"),
            names:[$r("app.string.pacing_latency"), $r("app.string.pacing_balanced"),
$r("app.string.pacing_balanced_alt"), $r("app.string.pacing_smoothness")],
            entries: ["latency", "balanced", "cap-fps", "smoothness"],
            value: "",
            onChange: this.onUpdateValue
          })
          CheckBoxPreference({
            pKey: "stretch_video",
            title: $r("app.string.title_checkbox_stretch_video"),
            summary: null,
            value: false,
            onChange: this.onUpdateValue
          })
        }.alignItems(HorizontalAlign.Start)
        Column({ space: 10 }) {
          Text($r("app.string.category_audio_settings"))
          ListPreference({
            pKey: "audio_config_list",
            title: $r("app.string.title_audio_config_list"),
            summary: $r("app.string.summary_audio_config_list"),
        names:[$r("app.string.audioconf_stereo"), $r("app.string.audioconf_51surround"),
$r("app.string.audioconf_71surround")],
        entries: ["2", "51", "71"],
        value: "1280x720",
        onChange: this.onUpdateValue
      })
    }.alignItems(HorizontalAlign.Start)
    Column({ space: 10 }) {
      Text($r("app.string.category_gamepad_settings"))
      SeekBarPreference({
        pKey: "seekbar_deadzone",
        title: $r("app.string.title_seekbar_deadzone"),
        summary: $r("app.string.summary_seekbar_deadzone"),
        value: "",
        onChange: this.onUpdateValue
      })
    }.alignItems(HorizontalAlign.Start)
    Column({ space: 10 }) {
      Text($r("app.string.category_input_settings"))
      CheckBoxPreference({
        pKey: "touchscreen_trackpad",
        title: $r("app.string.title_checkbox_touchscreen_trackpad"),
        summary: $r("app.string.summary_checkbox_touchscreen_trackpad"),
        value: false,
        onChange: this.onUpdateValue
      })
    }.alignItems(HorizontalAlign.Start)
    Column({ space: 10 }) {
      Text($r("app.string.category_on_screen_controls_settings"))
      CheckBoxPreference({
        pKey: "show_onscreen_controls",
        title: $r("app.string.title_checkbox_show_onscreen_controls"),
        summary: $r("app.string.summary_checkbox_show_onscreen_controls"),
        value: false,
        onChange: this.onUpdateValue
      })
      CheckBoxPreference({
        pKey: "vibrate_osc",
        title: $r("app.string.title_checkbox_vibrate_osc"),
        summary: $r("app.string.summary_checkbox_vibrate_osc"),
        value: false,
        onChange: this.onUpdateValue
      })
      CheckBoxPreference({
        pKey: "only_l3r3",
        title: $r("app.string.title_only_l3r3"),
        summary: $r("app.string.summary_only_l3r3"),
        value: false,
        onChange: this.onUpdateValue
      })
      CheckBoxPreference({
        pKey: "osc_opacity",
        title: $r("app.string.dialog_title_osc_opacity"),
        summary: $r("app.string.summary_osc_opacity"),
        value: false,
        onChange: this.onUpdateValue
      })
      CheckBoxPreference({
        pKey: "reset_osc",
        title: $r("app.string.title_reset_osc"),
        summary: $r("app.string.summary_reset_osc"),
        value: false,
        onChange: ()=>{
          router.pushUrl({url:"pages/VirtualControllerSettings"})
        }})
    }.alignItems(HorizontalAlign.Start)
    Column({ space: 10 }) {
      Text($r("app.string.category_host_settings"))
      CheckBoxPreference({
        pKey: "enable_sops",
        title: $r("app.string.title_checkbox_enable_sops"),
        summary: $r("app.string.summary_checkbox_enable_sops"),
        value: false,
        onChange: this.onUpdateValue
      })
      CheckBoxPreference({
        pKey: "host_audio",
        title: $r("app.string.title_checkbox_host_audio"),
        summary: $r("app.string.summary_checkbox_host_audio"),
        value: false,
        onChange: this.onUpdateValue
      })
    }.alignItems(HorizontalAlign.Start)
    Column({ space: 10 }) {
      Text($r("app.string.category_ui_settings"))
    }.alignItems(HorizontalAlign.Start)
    Column({ space: 10 }) {
      Text($r("app.string.category_advanced_settings"))
      CheckBoxPreference({
        pKey: "unlock_fps",
        title: $r("app.string.title_unlock_fps"),
        summary: $r("app.string.summary_unlock_fps"),
        value: false,
        onChange: this.onUpdateValue
      })
      CheckBoxPreference({
        pKey: "refresh_rate",
        title: $r("app.string.title_checkbox_reduce_refresh_rate"),
        summary: $r("app.string.summary_checkbox_reduce_refresh_rate"),
        value: false,
        onChange: this.onUpdateValue
      })
      CheckBoxPreference({
        pKey: "disable_warnings",
        title: $r("app.string.title_checkbox_disable_warnings"),
        summary: $r("app.string.summary_checkbox_disable_warnings"),
        value: false,
        onChange: this.onUpdateValue
      })
      ListPreference({
        pKey: "video_format",
        title: $r("app.string.title_video_format"),
            summary: $r("app.string.summary_video_format"),
            names:[$r("app.string.videoformat_auto"),  $r("app.string.videoformat_hevcalways"),
$r("app.string.videoformat_h264always")],
            entries: ["auto", "h265", "h264"],//$r("app.string.videoformat_av1always"), "av1",
            value: "h264",
            onChange: this.onUpdateValue
          })
          CheckBoxPreference({
            pKey: "enable_hdr",
            title: $r("app.string.title_enable_hdr"),
            summary: $r("app.string.summary_enable_hdr"),
            value: false,
            onChange: this.onUpdateValue
          })
          CheckBoxPreference({
            pKey: "full_range",
            title: $r("app.string.title_full_range"),
            summary: $r("app.string.summary_full_range"),
            value: false,
            onChange: this.onUpdateValue
          })
          CheckBoxPreference({
            pKey: "enable_perf_overlay",
            title: $r("app.string.title_enable_perf_overlay"),
            summary: $r("app.string.summary_enable_perf_overlay"),
            value: false,
            onChange: this.onUpdateValue
          })
          CheckBoxPreference({
            pKey: "enable_post_stream_toast",
            title: $r("app.string.title_enable_post_stream_toast"),
            summary: $r("app.string.summary_enable_post_stream_toast"),
            value: false,
            onChange: this.onUpdateValue
          })
        }.alignItems(HorizontalAlign.Start)
        Blank().height(30)
      }.padding(10)
    }.scrollable(ScrollDirection.Vertical)
  }.height('100%').backgroundColor($r("app.color.page_background"))
 }
}
@Builder
function BaseReference(hasCheck: Boolean = false, onClick: () => void) {
 Row() {
  Column({ space: 2 }) {
   Text(this.title).fontSize(20).fontColor(0xFFFFFF)
   Text(this.summary).fontSize(16).fontColor(0xC3C3C3)
  }.layoutWeight(1).alignItems(HorizontalAlign.Start)
  if (hasCheck) {
   Toggle({ type: ToggleType.Checkbox, isOn: this.value == true })
  }
 }.padding({ right: 20}).width('100%').onClick(onClick)
}
@CustomDialog
struct SeekBarDialog{
  @Link inputValue : number
  title: string| Resource = ""
  summary: Resource
  dialogController: CustomDialogController
  onConfirm: () => void
  build(){
    Column() {
      Text(this.title)
      Text(this.summary)
      Blank().height(50)
      Slider({
        value: this.inputValue,
        min: 0,
        max: 100,
        style: SliderStyle.InSet
      })
        .blockColor('#191970')
        .trackColor('#ffe0eaec')
        .selectedColor('#c3c3c3')
        .showTips(true)
        .onChange((value: number, mode: SliderChangeMode) => {
          this.inputValue = value
          console.info('value:' + value + 'mode:' + mode.toString())
        })
      Blank().height(30)
      Button('确定').onClick(()=>{
        this.onConfirm()
        this.dialogController.close()
      })
    }.padding(20)
  }
}
@Component
struct SeekBarPreference {
  @Consume  @Watch('onSettingsUpdated')
  settings: StreamSettings
  pKey: string
  title: Resource = null
  summary: Resource = null
  @State value: string = ""
  onChange: (key, value) => void
  aboutToAppear() {
  }
  onSettingsUpdated(){
    if(this.settings[this.pKey] != null){
      this.value = this.settings[this.pKey]
    }
  }
  dialogController: CustomDialogController = new CustomDialogController({
    builder: SeekBarDialog({
      inputValue: $value,
      title: this.title,
      summary: this.summary,
      onConfirm: ()=>{
        this.onChange(this.pKey, this.value)
      }
    }),
    autoCancel: true,
    alignment: DialogAlignment.Bottom,
    customStyle: false
  })
  build() {
    BaseReference(false, () => {
      this.dialogController.open()
      this.onChange(this.pKey, this.value)
    })
  }
}
@Component
struct ListPreference {
  @Consume  @Watch('onSettingsUpdated')
  settings: StreamSettings
  pKey: string
  title: Resource
  summary: Resource
  @State value: string = ""
  onChange: (key, value) => void
  private select: number = 0
  entries: string[] = []
  names: Resource[]
  private labels: string[] = []
  aboutToAppear() {
    this.getLabels()
  }
  onSettingsUpdated(){
    if(!this.settings || this.settings[this.pKey] == null){
      this.select == this.entries.indexOf(this.value)
    } else {
      this.value = this.settings[this.pKey]
      this.select = this.entries.indexOf(this.value)
    }
  }
  async getLabels() {
    this.onSettingsUpdated()
    for (let r of this.names) {
      this.labels.push(await getResString(this, r))
    }
  }
  build() {
    BaseReference(false, () => {
      TextPickerDialog.show({
        range: this.labels,
        selected: this.select,
        onAccept: (value: TextPickerResult) => {
          this.select = value.index
          this.value = this.entries[value.index]
          this.onChange(this.pKey, this.value)
        },
      })
    })
  }
}
```

```
@Component
struct CheckBoxPreference {
  @Consume  @Watch('onSettingsUpdated')
  settings: StreamSettings
  pKey: string
  title: Resource
  summary: Resource
  @State value: boolean = false
  onChange: (key, value) => void
  onSettingsUpdated(){
    if(this.settings[this.pKey] != null){
      this.value = this.settings[this.pKey]
    }
  }
  build() {
    BaseReference(true, () => {
      this.value = !this.value
      this.onChange(this.pKey, this.value)
    })
  }
}
```

Title.ets

```
import router from '@ohos.router'
@Component
export struct NavTitle {
  title: string | Resource
  build() {
    Stack({alignContent:Alignment.Start}){
      Row(){
        Text(this.title).fontColor(Color.White)
          .fontSize(23).textAlign(TextAlign.Center)
      }.
      width("100%")
      .justifyContent(FlexAlign.Center)
      .height(50)
      Icon({icon:$r('app.media.arrow_left')}).onClick(()=>{
        router.back()
      })
    }.width("100%").padding(10)
  }
}
@Component
export struct Icon {
  icon: Resource
  iconSize: number = 48
  build(){
    Image(this.icon)
      .width(this.iconSize).height(this.iconSize)
  }
}
@Component
export struct MainTitle {
  title: string
  build() {
    Row(){
      Icon({icon:$r("app.media.settings")}).onClick(()=>{
        router.pushUrl({ url:"pages/SettingsPage" })
      })
      Icon({icon: $r('app.media.ic_add_to_queue_white_48px')}).onClick(()=>{
        router.pushUrl({ url:"pages/AddPage" })
      })
    }.justifyContent(FlexAlign.SpaceBetween)
    .width("100%").padding(10)
    .height(50)
  }
}
```

DialogUtils.ets

Loading.ets

```
import emitter from '@ohos.events.emitter'
import { SpinnerEventId } from '../../uitls/CommonEvents'
@CustomDialog
export struct Spinner {
  title: string | Resource
  @State text: string = ""
  controller: CustomDialogController
  aboutToAppear(){
    let innerEvent = {
      eventId: SpinnerEventId,
    };
    emitter.on(innerEvent, (eventData)=>{
      this.text = eventData.data["message"]
    })
  }
  build() {
    Column(){
      Text(this.title).fontSize(18)
      Row(){
        LoadingProgress().height(100).width(80)
        if (this.text == ""){
          Text($r("app.string.conn_establishing_msg")).maxLines(2).height(100).textOverflow({overflow: 
TextOverflow.Ellipsis}).layoutWeight(1)
        } else {
          Text(this.text).maxLines(2).height(100).textOverflow({overflow: TextOverflow.Ellipsis}).layoutWeight(1)
        }
      }
    }.padding(20)
    .alignItems(HorizontalAlign.Start).backgroundColor(Color.White).borderRadius(10)
    .width("40%")
    .height(140)
  }
}
@CustomDialog
export struct Loading {
  title: string | Resource
  text: string | Resource = ""
  controller: CustomDialogController
  build() {
    Column(){
      Text(this.title).fontSize(18)
      Row(){
        LoadingProgress().height(100).width(80)
        Text(this.text).maxLines(2).height(100).textOverflow({overflow: TextOverflow.Ellipsis}).layoutWeight(1)
      }
    }.padding(20)
    .alignItems(HorizontalAlign.Start).backgroundColor(Color.White).borderRadius(10)
    .width("40%")
    .height(140)
  }
}
@CustomDialog
export struct Alert {
  title: string | Resource = "建立连接中"
  message: string | Resource
  controller: CustomDialogController
  build() {
    Column(){
      Text(this.title).fontSize(18)
      Blank().height(20)
      Text(this.message)
    }.padding(20)
    .alignItems(HorizontalAlign.Start).backgroundColor(Color.White).borderRadius(10)
    .width("40%")
  }
}
```

Index.ets

```
import hilog from '@ohos.hilog';
import router from '@ohos.router';
import LimelightCertProvider from '../entryability/crypto/LimelightCryptoProvider';
import { AddressTuple } from '../entryability/computers/ComputerDetails';
import { NvHttp } from '../entryability/http/NvHttp';
@Entry
@Component
struct Index {
  @State textValue: string = ''

  onCancel() {
    console.info('Callback when the first button is clicked')
  }
  onAccept() {
    console.info('Callback when the second button is clicked')
  }
  build() {
    Stack({ alignContent: Alignment.Top }){
      Row() {
        Button("start")
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
          .onClick(() => {
            router.pushUrl({ url:"pages/GamePage" })
          })
        Button("pair")
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
          .onClick(async  () => {
            try {
              const http = new NvHttp(new AddressTuple("192.168.3.5", 47989), 47984, null, LimelightCertProvider)
              const server = await http.getServerInfo(true)
              await http.pm.pair(server, "12345")
            }catch (e){
              console.log(e);
            }
          })
        Button("pc")
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
          .onClick(async  () => {
            try {
              router.pushUrl({ url:"pages/PcPage" })
            }catch (e){
              console.log(e);
            }
          })
      }
    }
    .width('100%')
    .height('100%').backgroundColor($r("app.color.page_background"))
  }
}
```

PcPage.ets

```
import { Icon} from './compoments/Title'
import { MainTitle } from './compoments/Title'
import { NavTitle } from './compoments/Title'
import viewModel from '../entryability/ComputerManagerViewModel'
import { ComputerState } from '../entryability/computers/ComputerDetails'
import { ComputerDetails} from '../entryability/computers/ComputerDetails'
import promptAction from '@ohos.promptAction'
import { PairState } from '../entryability/http/PairingManager'
import limelightCertProvider from '../entryability/crypto/LimelightCryptoProvider'
import { NvHttp } from '../entryability/http/NvHttp'
import router from '@ohos.router'
import { Alert } from './compoments/Loading'
async function getResString(com: any, r: Resource): Promise<string> {
  return await getContext(com).resourceManager.getStringValue(r)
}
@Component
struct PcView {
  @State detail: ComputerDetails = new ComputerDetails()
  onDelete: (ComputerDetails) => void
  @Builder
  pcMenu() {
    Menu() {
      if (this.detail.pairState == PairState.PAIRED) {
        MenuItem({ content: "浏览游戏列表" }).onClick(()=>{
          this.doAppList(this.detail, false, false)
        })
      } else {
        MenuItem({ content: "和电脑配对" }).onClick(()=>{
          this.doPair()
        })
      }
      MenuItem({ content: "测试网络连接" })
      MenuItem({ content: "删除电脑" }).onClick(() => {
        this.onDelete(this.detail)
      })
      MenuItem({ content: "查看详情" }).onClick((e) => {
        let alertDialog = new CustomDialogController({
          builder: Alert({
            title: "查看详情",
            message: this.detail.toString(),
          }),
          autoCancel: true,
          alignment: DialogAlignment.Center,
          customStyle: true
        })
        alertDialog.open()
      })
    }
  }
  click() {
    if (this.detail.state == ComputerState.ONLINE && this.detail.pairState != PairState.PAIRED) {
      this.doPair()
    } else if(this.detail.state == ComputerState.ONLINE) {
      this.doAppList(this.detail, false, false)
    } else {
    }
  }
  async doPair() {
    let message;
    let success = false;
    const computer = this.detail
    const httpConn = new NvHttp(this.detail.activeAddress,
      this.detail.httpsPort, null,
      limelightCertProvider);
    const state = await httpConn.fetchPairState()
    if (state == PairState.PAIRED) {
      message = null;
      success = true;
    } else {
      const pinStr = "12345";
      const dialogText =  await getResString(this, $r('app.string.pair_pairing_msg')) + " " + pinStr + "\n\n" + await
getResString(this, $r('app.string.pair_pairing_help'))
      let alertDialog = new CustomDialogController({
        builder: Alert({
          title: "配对中",
          message: dialogText,
        }),
        autoCancel: false,
        alignment: DialogAlignment.Center,
        customStyle: true
      })
      alertDialog.open()
      const pm = httpConn.pm
      const pairState = await pm.pair(await httpConn.getServerInfo(true), pinStr);
      if (pairState == PairState.PIN_WRONG) {
        message = await getResString(this, $r('app.string.pair_incorrect_pin'))
      }
      else if (pairState == PairState.FAILED) {
        if (computer.runningGameId != 0) {
          message = await getResString(this, $r('app.string.pair_pc_ingame'))
        }
        else {
          message = await getResString(this, $r('app.string.pair_fail'))
        }
      }
      else if (pairState == PairState.ALREADY_IN_PROGRESS) {
          message = await getResString(this, $r('app.string.pair_already_in_progress'))
      }
      else if (pairState == PairState.PAIRED) {
        message = null;
        success = true;
        this.detail.serverCert = true
        viewModel.runPoll(this.detail, false)
      }
      else {
        message = null;
      }
      alertDialog.close()
      alertDialog = undefined
      if (message)
        promptAction.showToast({ message: message })
      if (success){
        this.doAppList(computer, true, false);
      }
    }
  }
  doAppList(computer: ComputerDetails , newlyPaired:boolean,  showHiddenGames:boolean){
    router.pushUrl({url:"pages/AppPage", params: { uuid:  this.detail.uuid, computerName: this.detail.name,
rawAppList: this.detail.rawAppList}})
  }
  aboutToAppear() {
  }
  build() {
    Column() {
      Stack({ alignContent: Alignment.Center }) {
        Icon({ icon: $r('app.media.desktop_windows'), iconSize: 120 })
        if (this.detail.isLoading) {
          LoadingProgress().width(50).height(50).color(Color.White).offset({ y: -10 })
        } else {
          if (this.detail.state == ComputerState.ONLINE) {
            if (this.detail.pairState != PairState.PAIRED) {
              Icon({ icon: $r('app.media.baseline_lock'), iconSize: 48 }).offset({ y: -10 })
            }
          } else {
            Icon({ icon: $r('app.media.baseline_warning'), iconSize: 48 }).offset({ y: -10 })
          }
        }
      }
      Text(this.detail.name || "-").fontColor(Color.White)
    }.onClick(() => {
      this.click()
    }).bindContextMenu(this.pcMenu, ResponseType.LongPress)
  }
}
import taskpool from '@ohos.taskpool';
@Entry
```

```
@Component
struct PcPage {
  scroller: Scroller = new Scroller();
  @State pcList: ComputerDetails[] = []
  heightValue: number
  gridRowTemplate: string
  aboutToAppear() {
    viewModel.getComputerList().then((list) => {
      for(let d of list){
        d.isLoading = true
      }
      this.pcList = list
      viewModel.batchPollComputerList(list)
    })
    viewModel.onDetailsUpdate((news: ComputerDetails) => {
      var indexes = this.pcList.findIndex((d) => d.uuid == news.uuid)
      if (indexes < 0) {
        this.pcList.push(news)
      } else {
        this.pcList[indexes] = news
      }
      this.updateGrid()
    })
    this.updateGrid()
  }
  updateGrid() {
    var rows = Math.max(3, Math.round(this.pcList.length / 3))
    this.gridRowTemplate = '1fr '.repeat(rows);
    this.heightValue = rows * 192 - 8;
  }
  build() {
    Column() {
      MainTitle()
      Scroll(this.scroller) {
        Grid() {
          ForEach(this.pcList, (d) => {
            GridItem() {
              PcView({ detail: d, onDelete:(d)=>{
              }})
            }
          }, (item) => JSON.stringify(item))
        }.onKeyEvent((e)=>{
          console.log(e.keyCode+"");
        }).onMouse((e)=>{
          console.log(e.button+"");
        })
        .rowsTemplate(this.gridRowTemplate)
        .columnsTemplate('1fr 1fr 1fr')
        .height(this.heightValue)
      }.layoutWeight(1).scrollable(ScrollDirection.Vertical)
    }.padding(20).height("100%").width("100%").backgroundColor($r("app.color.page_background"))
  }
}
```

VirtualControllerSettings.ets

```
import { VirtualController, VirtualControllerBox } from '../virtual_controller/VirtualController'
@Entry
```

```
@Component
struct VirtualControllerSettings {
  virtualController = new VirtualController()
  aboutToAppear(){
  }
  build(){
    Stack({alignContent:Alignment.TopStart}){
      VirtualControllerBox({virtualController: this.virtualController, inputContext:this.virtualController.inputContext })
    }
  }
}
```

ControllerHandle.ts

```
import { NvConnection } from '../entryability/nvstream/NvConnection';
import { StreamSettings } from '../uitls/StreamSetttings';
import { GenericControllerContext } from './context/GenericControllerContext';
import { InputDeviceContext } from './context/GenericControllerContext';
export class ControllerHandle {
  conn: NvConnection
  prefConfig: StreamSettings
  currentControllers: number
  initialControllers: number
  defaultContext: InputDeviceContext = new InputDeviceContext()
  stickDeadzone: number
  inputDeviceContexts: Map<number, InputDeviceContext> = new Map();
  constructor(conn: NvConnection, prefConfig: StreamSettings) {
    this.conn = conn
    this.prefConfig = prefConfig
    let deadzonePercentage = parseInt(prefConfig.seekbar_deadzone);
    if (isNaN(deadzonePercentage) || deadzonePercentage <= 0) {
      deadzonePercentage = 1;
    }
    this.stickDeadzone = deadzonePercentage / 100.0;
    this.defaultContext.leftStickXAxis = 0; //MotionEvent.AXIS_X;
    this.defaultContext.leftStickYAxis = 1; // MotionEvent.AXIS_Y;
    this.defaultContext.leftStickDeadzoneRadius = this.stickDeadzone;
    this.defaultContext.rightStickXAxis = 11 //MotionEvent.AXIS_Z;
    this.defaultContext.rightStickYAxis = 14 //MotionEvent.AXIS_RZ;
    this.defaultContext.rightStickDeadzoneRadius = this.stickDeadzone;
    this.defaultContext.leftTriggerAxis = 23 // MotionEvent.AXIS_BRAKE;
    this.defaultContext.rightTriggerAxis = 22 // MotionEvent.AXIS_GAS;
    this.defaultContext.hatXAxis = 15 //MotionEvent.AXIS_HAT_X;
    this.defaultContext.hatYAxis = 16 // MotionEvent.AXIS_HAT_Y;
    this.defaultContext.controllerNumber = 0;
    this.defaultContext.assignedControllerNumber = true;
    this.defaultContext.external = false;
  }
  reportOscState(buttonFlags: number,
                 leftStickX: number, leftStickY: number,
                 rightStickX: number, rightStickY: number,
                 leftTrigger, rightTrigger: number) {
    const defaultContext = this.defaultContext
    defaultContext.leftStickX = leftStickX;
    defaultContext.leftStickY = leftStickY;
    defaultContext.rightStickX = rightStickX;
    defaultContext.rightStickY = rightStickY;
    defaultContext.leftTrigger = leftTrigger;
    defaultContext.rightTrigger = rightTrigger;
    defaultContext.inputMap = buttonFlags;
    this.sendControllerInputPacket(this.defaultContext);
  }
  private sendControllerInputPacket(originalContext: GenericControllerContext): void {
    const conn = this.conn
    const controllerNumber: number = originalContext.controllerNumber;
    let inputMap: number = 0;
    let leftTrigger: number = 0;
    let rightTrigger: number = 0;
    let leftStickX: number = 0;
    let leftStickY: number = 0;
    let rightStickX: number = 0;
    let rightStickY: number = 0;
    if (this.defaultContext.controllerNumber === controllerNumber) {
      inputMap |= this.defaultContext.inputMap;
      leftTrigger |= this.maxByMagnitude(leftTrigger, this.defaultContext.leftTrigger);
      rightTrigger |= this.maxByMagnitude(rightTrigger, this.defaultContext.rightTrigger);
      leftStickX |= this.maxByMagnitude(leftStickX, this.defaultContext.leftStickX);
      leftStickY |= this.maxByMagnitude(leftStickY, this.defaultContext.leftStickY);
      rightStickX |= this.maxByMagnitude(rightStickX, this.defaultContext.rightStickX);
      rightStickY |= this.maxByMagnitude(rightStickY, this.defaultContext.rightStickY);
    }
    if (originalContext.mouseEmulationActive) {
    } else {
      conn.sendControllerInput(
        controllerNumber,
        this.getActiveControllerMask(),
        inputMap,
        leftTrigger,
        rightTrigger,
        leftStickX,
        leftStickY,
        rightStickX,
        rightStickY
      );
    }
  }
  getActiveControllerMask(): number {
    return 1
  }
  maxByMagnitude(a: number, b: number): number {
    const absA = Math.abs(a);
    const absB = Math.abs(b);
    if (absA > absB) {
      return a;
    }
    else {
      return b;
    }
  }
}
```

VirtualController.ets

```
import { AnalogStick, } from './AnalogStick';
import { DigitalButton } from './DigitalButton';
import mediaquery from '@ohos.mediaquery';
import { ElementLayoutParam, VirtualControllerButton, VirtualControllerElement } from './common';
import List from '@ohos.util.List';
import { DigitalPad } from './DigitalPad';
import { createLeftStick } from './VirtualControllerConfigurationLoader';
import { VirtualControllerConfigurationLoader } from './VirtualControllerConfigurationLoader';
import Prompt from '@system.prompt';
import { ControllerHandle } from './ControllerHandle';
export enum ControllerMode {
  Active,
  MoveButtons,
  ResizeButtons
}
@Observed
export class ControllerInputContext {
  public inputMap: number = 0x0000;
  public leftTrigger: number = 0x00;
  public rightTrigger: number = 0x00;
  public rightStickX: number = 0x0000;
  public rightStickY: number = 0x0000;
  public leftStickX: number = 0x0000;
  public leftStickY: number = 0x0000;
}
export class VirtualController {
  controllerHandle: ControllerHandle
  currentMode: ControllerMode = ControllerMode.Active
  inputContext: ControllerInputContext = new ControllerInputContext();
  elements: List<VirtualControllerElement> = new List()
  constructor() {
  }
  addElement(element: VirtualControllerElement, x: number, y: number, width: number, height: number) {
    element.setLayout(new ElementLayoutParam(x, y, width, height))
    this.elements.add(element)
  }
  onSettingsClick(context: Context){
    let message = ""
    if (this.currentMode == ControllerMode.Active){
      this.currentMode = ControllerMode.MoveButtons;
      message = "Entering configuration mode (Move buttons)";
    } else if (this.currentMode == ControllerMode.MoveButtons) {
      this.currentMode = ControllerMode.ResizeButtons;
      message = "Entering configuration mode (Resize buttons)";
    } else {
      this.currentMode = ControllerMode.Active;
      VirtualControllerConfigurationLoader.saveProfile(this, context);
      message = "Exiting configuration mode";
    }
    Prompt.showToast({message})
  }
  sendControllerInputContext(){
    const inputContext = this.inputContext
    if(!this.controllerHandle)
      return null;
    this.controllerHandle.reportOscState(inputContext.inputMap,
      inputContext.leftStickX,
      inputContext.leftStickY,
      inputContext.rightStickX,
      inputContext.rightStickY,
      inputContext.leftTrigger,
      inputContext.rightTrigger)
  }
  setOpacity(opactiy: number) {
  }
}
@Builder
export function ShowVirtualController(){
  ForEach(this.virtualController.elements.convertToArray(), (d: VirtualControllerElement) => {
    VirtualControllerButton({ element: d, layout: d.layout })
  }, (d:VirtualControllerElement) =>d.elementId.toString())
  Button(){
    Text("setting")
  }.offset({y: 80}).onClick(()=>{
    this.virtualController.onSettingsClick(getContext(this))
  })
}
@Component
export struct VirtualControllerBox{
  virtualController: VirtualController
  @ObjectLink inputContext: ControllerInputContext
  aboutToAppear() {
    const loader = new VirtualControllerConfigurationLoader()
    loader.createDefaultLayout(this.virtualController)
    VirtualControllerConfigurationLoader.loadFromPreferences(this.virtualController, getContext(this))
  }
  build() {
    Stack({ alignContent: Alignment.TopStart }) {
      ShowVirtualController()
      Text(JSON.stringify(this.inputContext)).alignSelf(ItemAlign.Center).offset({x: 50,y:0})
    }.height('100%')
    .width('100%')
  }
}
DigitalButton.ets
import {
  drawLine,
  pressedColor,
  getPercent,
  getHeight,
  getWidth,
  drawCircle,
  getDefaultColor,
  VirtualControllerElement
} from './common';
import { VirtualController } from './VirtualController';
interface DigitalButtonListener {
  onClick(): void;
  onLongClick(): void;
  onRelease(): void;
}
export class DigitalButton extends VirtualControllerElement {
  private listeners: DigitalButtonListener[] = [];
  layer: number
  constructor(controller: VirtualController, elementId: number, layer: number) {
    super(controller, elementId)
    this.layer = layer
  }
  addDigitalButtonListener(listener: DigitalButtonListener): void {
    this.listeners.push(listener);
  }
  onElementTouchEvent(event: TouchEvent) {
    switch (event.type) {
      case TouchType.Down: {
        this.pressed = true
        this.onClickCallback();
        break;
      }
      case TouchType.Move: {
        break;
      }
      case TouchType.Up: {
        this.pressed = false
        this.onReleaseCallback()
        break;
      }
    }
  }
  onElementDraw(canvas: CanvasRenderingContext2D) {
    canvas.clearRect(0, 0, this.context.width, this.context.height)
    canvas.fillStyle = "#00000000"
    const strokeWidth = 2
    this.context.beginPath();
    this.context.lineWidth = strokeWidth;
    this.context.strokeStyle = this.pressed ? pressedColor : getDefaultColor(this.virtualController)
    this.context.ellipse(getPercent(this.context.width, 50), getPercent(this.context.height, 50),
getPercent(this.context.width, 50) - strokeWidth, getPercent(this.context.height, 50) - strokeWidth, 0, 0, Math.PI *
2);
    this.context.stroke()
    if (this.text) {
      this.context.textAlign = "center"
      this.context.fillStyle = this.pressed ? pressedColor : getDefaultColor(this.virtualController)
      const textSize = getPercent(this.context.width, 25)
      this.context.font = vp2px(textSize) + 'px sans-serif';
      this.context.fillText(this.text, getPercent(this.context.width, 50), getPercent(this.context.height, 65));
    }
  }
  text: string = "A"
  pressed: boolean = false
  private onClickCallback() {
    this.listeners.forEach(listener => listener.onClick());
  }
  private onLongClickCallback() {
    this.listeners.forEach(listener => listener.onLongClick());
  }
  private onReleaseCallback() {
    this.listeners.forEach(listener => listener.onRelease());
  }
}
DigitalPad.ets
import {
  drawLine,
  drawRect,
  pressedColor,
  getPercent,
  getHeight,
  getWidth,
  getDefaultColor,
  VirtualControllerElement
} from './common';
const DIGITAL_PAD_DIRECTION_NO_DIRECTION = 0;
export const DIGITAL_PAD_DIRECTION_LEFT: number = 1;
export const DIGITAL_PAD_DIRECTION_UP: number = 2;
export const DIGITAL_PAD_DIRECTION_RIGHT: number = 4;
export const DIGITAL_PAD_DIRECTION_DOWN: number = 8;
interface DigitalPadListener {
  onDirectionChange(direction:number);
}
export class DigitalPad extends VirtualControllerElement {
  pressed: boolean = false
  padDirection: number = DIGITAL_PAD_DIRECTION_NO_DIRECTION
  DPAD_MARGIN: number = 5;
  private listeners: DigitalPadListener[] = [];
  addDigitalPadListener(listener: DigitalPadListener): void {
    this.listeners.push(listener);
  }
  private newDirectionCallback(direction: number) {
    this.listeners.forEach(listener => listener.onDirectionChange(direction));
  }
  onElementTouchEvent(event: TouchEvent) {
    switch (event.type) {
      case TouchType.Down: {
        break;
      }
      case TouchType.Move: {
        let direction = 0;
        let x = event.changedTouches[0].x
        let y = event.changedTouches[0].y
        if (x < getPercent(getWidth(this.context), 33)) {
          direction |= DIGITAL_PAD_DIRECTION_LEFT;
        }
        if (x > getPercent(getWidth(this.context), 66)) {
          direction |= DIGITAL_PAD_DIRECTION_RIGHT;
        }
        if (y > getPercent(getHeight(this.context), 66)) {
          direction |= DIGITAL_PAD_DIRECTION_DOWN;
        }
        if (y < getPercent(getHeight(this.context), 33)) {
          direction |= DIGITAL_PAD_DIRECTION_UP;
        }
        this.padDirection = direction
        this.newDirectionCallback(direction);
        break;
      }
      case TouchType.Cancel:
      case TouchType.Up: {
        this.padDirection = 0;
        this.newDirectionCallback(this.padDirection);
        break;
      }
    }
  }
  onElementDraw(canvas: CanvasRenderingContext2D) {
    this.context.clearRect(0, 0, this.context.width, this.context.height)
    const strokeWidth = 2
    this.context.fillStyle = "#00000000"
    this.context.lineWidth = 2;
    if (this.padDirection == DIGITAL_PAD_DIRECTION_NO_DIRECTION) {
      this.context.strokeStyle = getDefaultColor(this.virtualController)
      drawRect(
        this.context,
        getPercent(getWidth(this.context), 36), getPercent(getHeight(this.context), 36),
        getPercent(getWidth(this.context), 63), getPercent(getHeight(this.context), 63),
      );
    }
    this.context.strokeStyle = (this.padDirection & DIGITAL_PAD_DIRECTION_LEFT) > 0 ? pressedColor :
getDefaultColor(this.virtualController)
    drawRect(
      this.context,
      strokeWidth + this.DPAD_MARGIN,
      getPercent(getHeight(this.context), 33),
      getPercent(getWidth(this.context), 33),
      getPercent(getHeight(this.context), 66)
    );
    this.context.strokeStyle = (this.padDirection & DIGITAL_PAD_DIRECTION_UP) > 0 ? pressedColor :
getDefaultColor(this.virtualController)
    drawRect(
      this.context,
      getPercent(getWidth(this.context), 33),
        strokeWidth + this.DPAD_MARGIN,
      getPercent(getWidth(this.context), 66),
        getPercent(getHeight(this.context), 33),
    );
    this.context.strokeStyle = (this.padDirection & DIGITAL_PAD_DIRECTION_RIGHT) > 0 ? pressedColor :
getDefaultColor(this.virtualController)
    drawRect(
      this.context,
      getPercent(getWidth(this.context), 66),
        getPercent(getHeight(this.context), 33),
      getWidth(this.context) - (strokeWidth + this.DPAD_MARGIN),
      getPercent(getHeight(this.context), 66),
    );
    this.context.strokeStyle = (this.padDirection & DIGITAL_PAD_DIRECTION_DOWN) > 0 ? pressedColor :
getDefaultColor(this.virtualController)
    drawRect(
      this.context,
      getPercent(getWidth(this.context), 33),
       getPercent(getHeight(this.context), 66),
      getPercent(getWidth(this.context), 66),
       getHeight(this.context) - (strokeWidth + this.DPAD_MARGIN),
    );
    this.context.strokeStyle = ((this.padDirection & DIGITAL_PAD_DIRECTION_LEFT) > 0 &&
    (this.padDirection & DIGITAL_PAD_DIRECTION_UP) > 0) ? pressedColor : getDefaultColor(this.virtualController)
    drawLine(this.context,
      strokeWidth + this.DPAD_MARGIN,
      getPercent(getHeight(this.context), 33),
      getPercent(getWidth(this.context), 33),
      strokeWidth + this.DPAD_MARGIN,
    )
    this.context.strokeStyle = ((this.padDirection & DIGITAL_PAD_DIRECTION_UP) > 0 &&
    (this.padDirection & DIGITAL_PAD_DIRECTION_RIGHT) > 0) ? pressedColor :
getDefaultColor(this.virtualController)
    drawLine(this.context,
      getPercent(getWidth(this.context), 66),
      strokeWidth + this.DPAD_MARGIN,
      getWidth(this.context) - (strokeWidth + this.DPAD_MARGIN),
      getPercent(getHeight(this.context), 33)
    )
    this.context.strokeStyle = ((this.padDirection & DIGITAL_PAD_DIRECTION_RIGHT) > 0 &&
    (this.padDirection & DIGITAL_PAD_DIRECTION_DOWN) > 0) ? pressedColor :
getDefaultColor(this.virtualController)
    drawLine(this.context,
      getWidth(this.context) - (strokeWidth + this.DPAD_MARGIN),
      getPercent(getHeight(this.context), 66),
      getPercent(getWidth(this.context), 66),
      getHeight(this.context) - (strokeWidth + this.DPAD_MARGIN),
    )
    this.context.strokeStyle = ((this.padDirection & DIGITAL_PAD_DIRECTION_DOWN) > 0 &&
    (this.padDirection & DIGITAL_PAD_DIRECTION_LEFT) > 0) ? pressedColor :
getDefaultColor(this.virtualController)
    drawLine(this.context,
      getPercent(getWidth(this.context), 33),
      getHeight(this.context) - (strokeWidth + this.DPAD_MARGIN),
      strokeWidth + this.DPAD_MARGIN,
      getPercent(getHeight(this.context), 66),
    )
  }
}
```

common.ets

```
import { VirtualController } from './VirtualController';
import { ControllerMode} from './VirtualController';
export function getPercent(value: number, percent: number): number {
  return value / 100 * percent;
}
export function getCorrectWidth(context: CanvasRenderingContext2D): number {
  return context.width > context.height ? context.height : context.width;
}
export function drawCircle(context: CanvasRenderingContext2D, x: number, y: number, radius: number) {
  context.beginPath();
  context.arc(x, y, radius, 0, Math.PI * 2, true);
  context.stroke()
}
export function drawRect(context: CanvasRenderingContext2D, left: number, top: number, right: number, bottom:
number, fill: boolean = false) {
  if (fill) {
    context.fillRect(left, top, right - left, bottom - top)
  } else {
    context.strokeRect(left, top, right - left, bottom - top)
  }
```

```
}
export function drawLine(context: CanvasRenderingContext2D, left: number, top: number, right: number, bottom:
number) {
  context.beginPath();
  context.moveTo(left, top);
  context.lineTo(right, bottom);
  context.stroke();
}
export function getWidth(context): number {
  return context.width
}
export function getHeight(context): number {
  return context.height
}
const normalColor = "#F0888888";
export const pressedColor = "#F00000FF";
const configMoveColor = "#F0FF0000";
const configResizeColor = "#F0FF00FF";
const configSelectedColor = "#F000FF00";
export function getDefaultColor(virtualController: VirtualController) {
  if (virtualController.currentMode == ControllerMode.MoveButtons)
    return configMoveColor;
  else if (virtualController.currentMode == ControllerMode.ResizeButtons)
    return configResizeColor;
  else
    return normalColor;
}
enum Mode {
  Normal,
  Resize,
  Move
}
@Observed
export class ElementLayoutParam {
  x: number;
  y: number;
  width: number;
  height: number;
  constructor(x: number, y: number, width: number, height: number) {
    this.x = x
    this.y = y
    this.width = width
    this.height = height
  }
}
export abstract class VirtualControllerElement {
  public static EID_DPAD = 1;
  public static EID_LT = 2;
  public static EID_RT = 3;
  public static EID_LB = 4;
  public static EID_RB = 5;
  public static EID_A = 6;
  public static EID_B = 7;
  public static EID_X = 8;
  public static EID_Y = 9;
  public static EID_BACK = 10;
  public static EID_START = 11;
  public static EID_LS = 12;
  public static EID_RS = 13;
  public static EID_LSB = 14;
  public static EID_RSB = 15;
  elementId: number
  virtualController: VirtualController
  context: CanvasRenderingContext2D
  normalColor = "#F0888888";
  pressedColor = "#F00000FF";
  private configMoveColor = "#0xF0FF0000";
  private configResizeColor = "#0xF0FF00FF";
  private configSelectedColor = "#0xF000FF00";
  protected startSize_x: number;
  protected startSize_y: number;
  position_pressed_x: number = 0;
  position_pressed_y: number = 0;
  private currentMode = Mode.Normal;
  layout: ElementLayoutParam
  constructor(controller: VirtualController, elementId: number) {
    this.virtualController = controller;
    this.elementId = elementId;
  }
  setLayout(layout: ElementLayoutParam) {
    this.layout = layout;
  }
  getWidth(): number {
    return this.context.width
  }
  getHeight(): number {
    return this.context.height
  }
  getDefaultStrokeWidth(): number {
    return 2
  }
  resizeElement: (pressed_x, pressed_y, width, height) => void = (pressed_x, pressed_y, width, height) => {
    const layoutParams = this.layout
    let newHeight = height + (this.startSize_y - pressed_y);
    let newWidth = width + (this.startSize_x - pressed_x);
    layoutParams.height = newHeight > 20 ? newHeight : 20;
    layoutParams.width = newWidth > 20 ? newWidth : 20;
  }
  moveElement: (pressed_x, pressed_y, x, y) => void = (pressed_x, pressed_y, x, y) => {
    const layoutParams = this.layout
    const dx = x - pressed_x;
    const dy = y - pressed_y;
    layoutParams.x += dx
    layoutParams.y += dy
    this.position_pressed_x += dx;
    this.position_pressed_y += dy;
  }
  abstract onElementDraw(canvas: CanvasRenderingContext2D);
  abstract onElementTouchEvent(event: TouchEvent)
  onDraw(canvas: CanvasRenderingContext2D) {
    this.context = canvas
    this.onElementDraw(canvas);
```

```
    if (this.currentMode != Mode.Normal) {
      canvas.strokeStyle = configSelectedColor
      const strokeWidth = this.getDefaultStrokeWidth()
      canvas.lineWidth = strokeWidth
      drawRect(canvas, strokeWidth, strokeWidth, getWidth(canvas) - strokeWidth, getHeight(canvas) - strokeWidth)
    }
  }
  actionEnableMove() {
    this.currentMode = Mode.Move;
  }
  actionEnableResize() {
    this.currentMode = Mode.Resize;
  }
  actionCancel() {
    this.currentMode = Mode.Normal;
  }
  getConfiguration():string {
    return JSON.stringify(this.layout)
  }
  loadConfiguration(configuration: string){
    Object.assign(this.layout, JSON.parse(configuration))
  }
  onSizeChanged(canvas: CanvasRenderingContext2D) {
  }
  touchId: number = -1
  onTouchEvent(event: TouchEvent) {
    if (this.virtualController.currentMode == ControllerMode.Active) {
      return this.onElementTouchEvent(event);
    }
    let touch = null
    if(this.touchId > -1){
      touch = event.changedTouches.find((t)=>t.id == this.touchId)
      if (touch == null)
        return;
    }
    switch (event.type) {
      case TouchType.Down: {
        if(this.touchId == -1){
          touch = event.changedTouches[0]
          this.touchId = touch.id
        }
        this.position_pressed_x = touch.x;
        this.position_pressed_y = touch.y;
        this.startSize_x = this.getWidth();
        this.startSize_y = this.getHeight();
        if (this.virtualController.currentMode == ControllerMode.MoveButtons)
          this.actionEnableMove();
        else if (this.virtualController.currentMode == ControllerMode.ResizeButtons)
          this.actionEnableResize();
        return true;
      }
      case TouchType.Move: {
        switch (this.currentMode) {
          case Mode.Move: {
            this.moveElement(
              this.position_pressed_x,
              this.position_pressed_y,
              touch.x,
              touch.y);
            break;
          }
          case Mode.Resize: {
            this.resizeElement(
              this.position_pressed_x,
              this.position_pressed_y,
              touch.x,
              touch.y);
            break;
          }
          case Mode.Normal: {
            break;
          }
        }
        return true;
      }
      case TouchType.Cancel:{}
      case TouchType.Up: {
        this.touchId = -1
        this.actionCancel();
        return true;
      }
      default: {
      }
    }
    return true;
  }
 }
 @Component
 export struct VirtualControllerButton {
  private settings: RenderingContextSettings = new RenderingContextSettings(true)
  private context: CanvasRenderingContext2D = new CanvasRenderingContext2D(this.settings)
  element: VirtualControllerElement
  @ObjectLink layout: ElementLayoutParam
  onLayout() {
    this.element.onSizeChanged(this.context)
  }
  build() {
    Canvas(this.context)
      .height(this.layout.height)
      .width(this.layout.width)
      .translate({ x: this.layout.x, y: this.layout.y })
      .opacity(1)
      .onReady(() => {
        this.element.onDraw(this.context)
      })
      .onTouch((e) => {
        this.element.onTouchEvent(e)
        this.element.onDraw(this.context)
      })
  }
 }
```