MediaMesh Shared Memory SDK

Overview of MediaMesh Shared Memory

Local Shared Memory. Applications and containers running on the same host can read/write video, audio and timed metadata to shared memory using the *MediaMesh Shared Memory SDK* documented below in sections 1-7 below.

Global Shared Memory. Once a video, audio or timed metadata stream is available in local shared memory, applications can access that stream from anywhere, globally, by calling the *MediaMesh Remote Connection Service* with the *objectID* of that media stream (obtained from the *MediaMesh Object Service*). This access spans cloud Availability Zones (AZs), cloud Regions, cloud accounts, VPCs and even on-premise data centers. Section 3.5 of the MediaMesh Developer's Guide documents how to use the MediaMesh Remote Connection Service.

Therefore, the MediaMesh shared memory SDK (documented below) not only enables inter-process communication on a single machine, it is also foundational to global shared memory.

1. MediaMesh Shared Memory SDK

The MediaMesh Shared Memory SDK is accessible at https://github.com/tvunetworks-x/tvu-shared-memory. It includes a binary, libshmmediawrap-x86_64-*version*, which supports the function calls described in the following sections of the document.

- Section 2 describes calls supporting read/write of video, audio or timed metadata with ***constant item size***. These calls begin with the prefix ***LibShm*** .
- Section 3 describes calls supporting read/write of video, audio or timed metadata with a ***variable item size***. These calls begin with the prefix ***LibViShm*** ("Vi" stands for "variable item").
- Section 4 describes additional calls useful when reading/writing ***timed-metadata***.
- Section 5 documents command sequences for typical shared memory operations.
- Appendices A and B document key structures and supported media types

The GitHub site also contains sample code for shared memory reads and writes.

2. *LibShm,* shared memory APIs for constant sized items

2.1 Creating a Shared Memory Handle

Function: `LibShmMediaCreate`

Creates (or opens if it exists) a SHM handle for **writing**.

```
libshm_media_handle_t LibShmMediaCreate(const char
*pMemoryName, uint32_t header_len, uint32_t item_count,
uint32_t item_length);
```

- `pMemoryName`: Name of SHM.
- `header_len`: Size reserved for header/meta info.
- `item_count`: Number of items.
- `item_length`: Size of each item.

Returns: A SHM handle or `NULL` if failed.

2.2 Opening a Shared Memory Handle for Reading

Function: `LibShmMediaOpen`

Opens an existing SHM for **reading**.

```
libshm_media_handle_t LibShmMediaOpen(const char *pMemoryName,
libshm_media_readcb_t cb, void *opaq);
```

- `pMemoryName`: Name of SHM entry
- Takes a callback (`cb`) for read events.
- `opaq` is user-provided opaque data.

Returns: A SHM handle or `NULL` if failed.

2.3 Checking If SHM is Sendable

Function: `LibShmMediaPollSendable`

Polls if SHM is ready to send data (non-blocking wait).

```
int LibShmMediaPollSendable(libshm_media_handle_t h, unsigned
int timeout);
```

- `h`: Shared memory handle.
- `timeout`: poll's timeout value in milliseconds

Returns `>0` if ready, `0` if not ready, `<0` on error (need to destroy & create the handle again)

2.4 Sending Data

Function: `LibShmMediaSendData`

Sends media data into SHM.

```
int LibShmMediaSendData(libshm_media_handle_t h, const
libshm_media_head_param_t *pmh, const libshm_media_item_param_t
*pmi);
```

- `h`: Shared memory handle.
- `pmh`: input head information.
- `pmi`: input data information

Returns `>0` send success, size of write, `0` if not ready, `<0` I/O error

---

2.5 Reading Data

Function: `LibShmMediaPollReadData`

Polls and **reads** media data from SHM

```
int LibShmMediaPollReadData(libshm_media_handle_t h,
libshm_media_head_param_t *pmh, libshm_media_item_param_t *pmi,
unsigned int timeout);
```

- `h`: Shared memory handle
- `pmh`: Destination header information structure.
- `pmi`: Destination data information structure
- `timeout`: poll's timeout value in milliseconds

Returns `>0` if success, `0` if not ready (wait & try again), `<0` if failure

2.6 Destroying SHM

Function: `LibShmMediaDestroy`

Cleans up and **destroys** a SHM handle.

```
void LibShmMediaDestroy(libshm_media_handle_t h);
```

- `h`: Shared memory handle

---

3. LibViShm, Shared Memory APIs for Variable Sized Items

For SHMs where **items are variable-sized**. (The “Vi” in these function calls stands for “variable item”).

3.1 Creating a Variable SHM Handle

Function: `LibViShmMediaCreate`

Creates (or opens if it exists) a SHM handle for **writing** variable sized data. Unlike the similar call for constant sized item shared memory, this function takes as an argument the *total size* of a shared memory rather than the *item size*.

```
libshm_media_handle_t LibViShmMediaCreate(const char
*pMemoryName, uint32_t header_len, uint32_t item_count,
uint64_t total_size);
```

- `pMemoryName`: Name of SHM.
- `header_len`: Size reserved for header/meta info.
- `item_count`: Number of items.
- `total_size`: total size of SHM

Returns: A SHM handle or `NULL` if failed.

3.2 Opening Variable SHM for Reading

Function: `LibViShmMediaOpen`

Opens an existing variable item SHM for **reading**.

```
libshm_media_handle_t LibViShmMediaOpen(const char
*pMemoryName, libshm_media_readcb_t cb, void *opaq);
```

- `pMemoryName`: Name of SHM entry
- Takes a callback (`cb`) for read events.
- `opaq` is user-provided opaque data

Returns: A SHM handle or `NULL` if failed.

3.3 Polling Sendability

Function: `LibViShmMediaPollSendable`

This behaves like the corresponding call for constant SHM, but for variable sized item SHM. **Polls** if SHM is ready to send data (non-blocking wait).

```
int LibShmViMediaPollSendable(libshm_media_handle_t h, unsigned
int timeout);
```

- `h`: Shared memory handle.
- `timeout`: poll's timeout value in milliseconds

Returns `>0` if ready, `0` if not ready, `<0` I/O error (need to destroy/create the handle again)

3.4 Sending Data

Function: `LibViShmMediaSendData`

Sends media variable item sized data into SHM.

```
int LibShmViMediaSendData(libshm_media_handle_t h, const
libshm_media_head_param_t *pmh, const libshm_media_item_param_t
*pmi);
```

- `h`: Shared memory handle.
- `pmh`: input header structure information.
- `pmi`: input data structure information

Returns `>0` send success, size of write, `0` if not ready, `<0` I/O error.

3.5 Destroying Variable SHM

Function: `LibViShmMediaDestroy`

```
void LibViShmMediaDestroy(libshm_media_handle_t h);
```

- `h`: Shared memory handle

Cleans up and **destroys** a SHM handle.

---

4. Metadata Extension APIs

MediaMesh supports **user-defined metadata** to extend shared memory capabilities. **Note**: To ensure interoperability and proper testing, metadata extensions must be approved by TVU Networks.

4.1 Estimating Buffer Size

Function: `LibShmMediaEstimateExtendDataSize`

Determines how much buffer space is needed for metadata.

```
int LibShmMediaEstimateExtendDataSize(/*IN*/const
libshmmedia_extend_data_info_t* pExtendData);
```

- `pExtendData[IN]`: source external data information structure

Returns `>0` buffer size, `0` no buffer needed, `<0` failed

4.2 Writing Extended Data

Function: `LibShmMediaWriteExtendData`

Writes external metadata into a buffer..

```
int LibShmMediaWriteExtendData(/*OUT*/uint8_t dataBuffer[],
/*IN*/int bufferSize, /*IN*/const
libshmmedia_extend_data_info_t* pExtendData);
```

- `dataBuffer[OUT]`: Destination buffer
- `buffersize[IN]`: Destination buffer size
- `pExtendData[IN]`: source external data information structure

Returns `>=0` actual size of buffer write, `<0` write failure

---

5. Typical Workflows

The examples below outline the sequence of LibShm commands for typical workflows.

5.1 Writing Data to SHM

1. **Create** SHM: call `LibShmMediaCreate` to obtain shared memory handle
2. **Poll** sendable: call `LibShmMediaPollSendable` to determine if the shared memory handle is ready to send.
3. **Send** data: Once the shared memory handle is ready (step 2 above), call `LibShmMediaSendData` to send data to shared memory.
4. **Destroy** SHM if unused: If the shared media handle is no longer needed, call `LibShmMediaDestroy` to destroy it.

5.2 Reading Data from SHM

1. **Open** SHM: call `LibShmMediaOpen` to open the shared memory and obtain the shared memory handle.
2. **Poll and read**: call `LibShmMediaPollReadData` to read data from shared memory.
3. **Destroy** SHM after use: call `LibShmMediaDestroy` if the handle is no longer needed to destroy it.

5.3 Writing Timed Metadata (such as SCTE, CC, timecode, etc)

1. Define a `libshmmedia_extend_data_info_t` object as myExt.
2. Initialize the object and set the desired fields, carefully evaluating the length and pointers of structure members.
3. Call `LibShmMediaEstimateExtendDataSize` to obtain the buffer size.
4. Allocate buffer according to the size returned by step #3.
5. Call `LibShmMediaWriteExtendData` to write metadata to the buffer.

6. For the members `p_userData` and `i_userDataLen` of `libshm_media_item_param_t`, verify that the data you want to attach is consistent with these values, and set the `i_userDataType` field explicitly to the constant `LIBSHM_MEDIA_TYPE_TVU_EXTEND_DATA_V2`.
7. Call `LibShmMediaSendData` to write the contents of the buffer (metadata) as well as associated video and audio to shared memory.

5.4 Parsing Timed Metadata (such as SCTE, CC, timecode, etc.)

1. Define a `libshmmedia_extend_data_info_t` object as myExt.
2. Initialize the object to zero.
3. call LibShmMeidaParseExtendData to parse the extension data point to libshmmedia_extend_data_info_t structure.

Appendix A

A.1 Media Head Structure

Describes media-level parameters (dimensions, codecs, sample rate, etc).

```
typedef struct SLibShmMediaHeadParam
{
    uint32_t        u_structSize; // struct size for extension compatibility
    int32_t         i_vbr;      // can be ignored to set.
    int32_t         i_sarw;     // codec sarw (pixel width ratio)
    int32_t         i_sarh;     // codec sarh (pixel height ratio)
    int32_t         i_srcw;     // source codec width
    int32_t         i_srch;     // source codec height
    int32_t         i_dstw;     // output YUV width
    int32_t         i_dsth;     // output YUV height
    uint32_t        u_videofourcc;
    int32_t         i_duration;
    int32_t         i_scale;
    uint32_t        u_audiofourcc;
    int32_t         i_channels;
    int32_t         i_depth;
    int32_t         i_samplerate;
}libshm_media_head_param_t;
```

A.2 Media Item Structure

Describes an individual media item (frames, timestamps, metadata).
The details can be seen as Appendix C(video), Appendix D(audio), Appendix E(meta data).

```
typedef struct SLibShmMediaItemParam
{
     uint32_t        u_structSize; // struct size for extension compatibility
    int         i_totalLen; // can be unset
    const uint8_t   *p_vData; // video's data point
    int         i_vLen; // video frame len
    int64_t     i64_vpts; // video frame pts
    int64_t     i64_vdts; // video frame dts
    int64_t     i64_vct; // video frame's create time.
    const uint8_t    *p_aData; //audio data point.
    int         i_aLen; // audio length
    int64_t     i64_apts; // audio frame pts
    int64_t     i64_adts; // audio frame dts
    int64_t     i64_act; // audio frame create time.
    const uint8_t     *p_sData; // subtitle data point
```

```
    int         i_sLen; // subtitle data length
    int64_t     i64_spts; // subtitle pts
    int64_t     i64_sdts; // subtitle dts
    int64_t     i64_sct; // subtitle create time
    const uint8_t    *p_CCData; // closed caption data point
    int         i_CCLen; // closed caption data length
    const uint8_t    *p_timeCode; // timecode data point
    int         i_timeCode; // timecode data length
    uint32_t    u_frameType; // frame type
    uint32_t    u_picType; // picture type
    const uint8_t    *p_userData; // metadata
    int         i_userDataLen; // metadata length
    int64_t     i64_userDataCT;// meta data create time.
    int         i_userDataType; // user data type
    uint32_t    i_interlaceFlag; // interlace flag. 1-progressive,
2-interlace
     uint32_t    u_copied_flags; //shows the status of write actions by
datatype

#define LIBSHM_MEDIA_VIDEO_COPIED_FLAG    0x01
#define LIBSHM_MEDIA_AUDIO_COPIED_FLAG    0x02
#define LIBSHM_MEDIA_SUBTITLE_COPIED_FLAG 0x04
#define LIBSHM_MEDIA_EXT_COPIED_FLAG      0x08
#define LIBSHM_MEDIA_CLOSED_CAPTION_COPIED_FLAG   0x08
#define LIBSHM_MEDIA_USER_DATA_COPIED_FLAG   0x10

    uint32_t    u_read_index;
    void                                *p_opaq;
    libshm_media_process_handle_t       h_media_process;
}libshm_media_item_param_t;
```

Appendix B: MediaMesh Supported Media Types

MediaMesh supported video and multimedia file types are defined by the following four character code (FourCC) identifiers:

Video Formats are defined as `ETVUPixfmtVideoFourCC` in `libtvu_media_fourcc.h`

```
K_TVU_PIXFMT_VIDEO_FOURCC_YUV420P   = _TVU_LE_FOURCCTAG('I', '4', '2', '0'),
K_TVU_PIXFMT_VIDEO_FOURCC_YUYV422   = _TVU_LE_FOURCCTAG('V', '4', '2', '2'),
K_TVU_PIXFMT_VIDEO_FOURCC_RGB24     = _TVU_LE_FOURCCTAG('R', 'G', 'B', 24 ),
K_TVU_PIXFMT_VIDEO_FOURCC_BGR24     = _TVU_LE_FOURCCTAG('B', 'G', 'R', 24 ),
K_TVU_PIXFMT_VIDEO_FOURCC_YUV422P   = _TVU_LE_FOURCCTAG('I', '4', '2', '2'),
K_TVU_PIXFMT_VIDEO_FOURCC_YUV444P   = _TVU_LE_FOURCCTAG('I', '4', '4', '4'),
K_TVU_PIXFMT_VIDEO_FOURCC_YUV410P   = _TVU_LE_FOURCCTAG('I', '4', '1', '0'),
K_TVU_PIXFMT_VIDEO_FOURCC_YUV411P   = _TVU_LE_FOURCCTAG('I', '4', '1', '1'),
K_TVU_PIXFMT_VIDEO_FOURCC_GRAY8     = _TVU_LE_FOURCCTAG('G', 'R', 'E', 'Y'),
K_TVU_PIXFMT_VIDEO_FOURCC_MONOWHITE = _TVU_LE_FOURCCTAG('B', '1', 'W', '0'),
K_TVU_PIXFMT_VIDEO_FOURCC_MONOBLACK = _TVU_LE_FOURCCTAG('B', '0', 'W', '1'),
K_TVU_PIXFMT_VIDEO_FOURCC_PAL8      = _TVU_LE_FOURCCTAG('P', 'A', 'L', '8'),
K_TVU_PIXFMT_VIDEO_FOURCC_YUVJ420P  = _TVU_LE_FOURCCTAG('J', '4', '2', '0'),
K_TVU_PIXFMT_VIDEO_FOURCC_YUVJ422P  = _TVU_LE_FOURCCTAG('J', '4', '2', '2'),
K_TVU_PIXFMT_VIDEO_FOURCC_YUVJ444P  = _TVU_LE_FOURCCTAG('J', '4', '4', '4'),
K_TVU_PIXFMT_VIDEO_FOURCC_UYVY422   = _TVU_LE_FOURCCTAG('U', 'Y', 'V', 'Y'),
```

Audio Formats are defined as `ETVUPixfmtAudioFourCC` in `libtvu_media_fourcc.h`

```
K_TVU_AUDIO_FOURCC_WAVE_48K_16      = _TVU_LE_FOURCCTAG('W', 'A', 'V', '1')
```

Appendix C: MediaMesh Video Parameters Detail

MediaMesh video programming parameter as the Appendix A.
There are some explanations for the video parameters at SLibShmMediaHeadParam structure, as

i_dstw : the destination resolution width

i_dsth : the destination resolution height

u_videofourcc : the video data store format, see the detail as Appendix B.

i_duration : the video stream fps(frame rate per second) denominator, it is usually 1001 for N video(NTSC), and it would be 1000 for P video(PAL).

i_scale : the video stream fps(frame rate per second) numerato, it is usually 30000/60000 for N video(NTSC), and it would be 25000/50000 for PAL.

Appendix D: MediaMesh Audio Parameters Detail

MediaMesh audio programming parameter as the Appendix A.
There are some explanations for the audio parameters at SLibShmMediaHeadParam structure, as

i_channels : the audio channel layout is 32bit which used to express the audio track/channel status. In fact every 4bits expresses the channel number of one track, so it supports a maximum of 8 tracks, as channel layout is 0x22222222, which means 8 tracks stereo. However, to support 16 track mono, you can just use 0xFFFF10 to express 16 track mono.

i_depth : the audio sample data depth, it is usually 16bits depth here.

i_samplerate : the audio sample rate, it is usually 48000 here.

u_audiofourcc : the audio data store format, see the detail as Appendix B, it is usually PCM Signed 16 & sample rate 48000.

Appendix E: MediaMesh Metadata Parameters Detail

MediaMesh meta data programming parameter is defined as structure libshmmedia_extend_data_info_t at file libshm_media_extension_protocol.h.

There are some explanations for the meta data items, as

p_uuid_data/i_uuid_length : tvu's uuid which is used to express the source unique identifier. However you can ignore it in fact.

p_cc608_cdp_data/i_cc608_cdp_length : they are used to store the CDP of CEA708/EIA608. You can ignore it if the stream did not have it or you do not need it.

p_scte104_data/i_scte104_data_len : they are used to store the raw SCTE message as from scte35 parsing of MPEGTS stream.

p_timecode/i_timecode : they are used to store the timecode of TVU's format, as hh::mm::ss.frame. However you can ignore it.

p_metaDataPts/i_metaDataPts : they are used to store the timestamp of metadata, the i_metaDataPts is constant 8, the p_metaDataPts is little endian of 64bits integer.

```
#include "libshm_media.h"
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>
#include <inttypes.h>

#define TVUUTIL_GET_SYS_MS64()
_xxtvuutil_get_sys_ms64()
#define VIDEO_FRAME_SIZE                          10240
#define TVU_LINUX 1

static inline
int64_t _xxtvuutil_get_sys_ms64()
{
#ifdef TVU_WINDOWS
    int64_t tmNow = 0;
    struct _timeb timebuffer;
    _ftime_s(&timebuffer);
    tmNow = timebuffer.time;
    tmNow *= 1000;
    tmNow += timebuffer.millitm;
    return tmNow;
#elif defined(TVU_MINGW)
    int64_t tmNow = 0;
    struct _timeb timebuffer;
    _ftime(&timebuffer);
    tmNow = timebuffer.time;
    tmNow *= 1000;
    tmNow += timebuffer.millitm;
    return tmNow;
#elif defined(TVU_LINUX)
    int64_t tmNow = 0;
    struct timeval  tv;
    gettimeofday(&tv, NULL);
    tmNow = tv.tv_sec * 1000 + tv.tv_usec/1000;
    return tmNow;
#endif
}

static void Sleep(uint32_t ms)
```

```
{
    usleep(ms*1000);
}

static bool g_exit = false;

static int WriteSampleCode(const char *name, int count, int
item_size)
{
    unsigned int                i       = 0;
    libshm_media_handle_t   h           = NULL;
    libshm_media_head_param_t   ohp;
    {
        memset(&ohp, 0, sizeof(ohp));
        ohp.u_structSize = sizeof(libshm_media_head_param_t);
    }
    libshm_media_item_param_t   ohi;
    {
        memset(&ohi, 0, sizeof(ohi));
        ohi.u_structSize = sizeof(libshm_media_item_param_t);
    }

    int64_t                     now     = 0;
    int64_t                     base    = 0;
    int64_t                     next_pts= 0;

    h       = LibShmMediaCreate(name, 1024, count, item_size); //
create shared memory handle with head size, item count, item size

    if (!h)
    {
        printf("create libshm media handle failed\n");
        return -1;
    }

    printf("shm version : %x\n", LibShmMediaGetVersion(h));
    fflush(stdout);

    ohp.i_dstw          = 1920;
    ohp.i_dsth          = 1080;
```

```
    ohp.u_videofourcc   = 'h' << 24 | '2' << 16 | '6' << 8  |
'4';
    ohp.i_duration      = 1001;
    ohp.i_scale         = 30000;
    ohp.u_audiofourcc   = 'a' << 24 | 'a' << 16 | 'c' << 8 | '0';
    ohp.i_channels      = 2;
    ohp.i_depth         = 16;
    ohp.i_samplerate    = 48000;

    now = TVUUTIL_GET_SYS_MS64();
    base = now;

    uint8_t *video_data = (uint8_t *)malloc(item_size);

    memset((void *)video_data, 'v', item_size);

    while (!g_exit) {
        now = TVUUTIL_GET_SYS_MS64();
        next_pts    = base + i * 33;

        if (now < next_pts)
        {
            Sleep(1);
            continue;
        }

        /* prepare video/audio data. */
        ohi.i64_vdts    =
        ohi.i64_vpts    = next_pts;
        ohi.p_vData     = (uint8_t *)video_data;
        ohi.i_vLen      = item_size;
        ohi.i64_apts    =
        ohi.i64_adts    = next_pts;
        ohi.p_aData     = (uint8_t *)"a1a2a3a4a5";
        ohi.i_aLen      = 10;

        int ret = 0;
        uint8_t aExtBuff[1024] = {0};
        uint32_t iExtBuffSize = 0;
        char uuid_str[37] =
"1234567890abcdefghijklfmopqrstuvwxyz";
```

```
        uint8_t cc608[73] = {0};
        memset(cc608, '6', 72);


        /* prepare extension data. */
        libshmmedia_extend_data_info_t myExt;
        {
            memset(&myExt, 0, sizeof (myExt));
        }

        {
            myExt.p_caption_text = cc608;
            myExt.i_cc608_cdp_length = 72;
        }

        {
            myExt.p_uuid_data = (const uint8_t *)uuid_str;
            myExt.i_uuid_length = strlen(uuid_str);
        }

        {
            uint32_t timecode = 0;
            myExt.p_timecode = (const uint8_t*)&timecode;
            myExt.i_timecode = sizeof (uint32_t);
        }

        int iExtBuffSizeBeforeAlloc =
LibShmMediaEstimateExtendDataSize(&myExt);
        iExtBuffSize = LibShmMediaWriteExtendData(aExtBuff,
iExtBuffSizeBeforeAlloc, &myExt);

        ohi.i_userDataType =
LIBSHM_MEDIA_TYPE_TVU_EXTEND_DATA_V2; // extension data need this
type.
        ohi.p_userData = aExtBuff;
        ohi.i_userDataLen = iExtBuffSize;

        ret     = LibShmMediaPollSendable(h, 0);

        if (ret < 0) {
            printf("sending failed\n");
```

```
            break;
        } else if (ret == 0) {
            printf("need wait to resend\n");
            Sleep(1);
            continue;
        }
        else {
            // poll success
            ret = LibShmMediaSendData(h, &ohp, &ohi); // write
the item data to shared memory.
            if (ret < 0) {
                printf("sendable status, send failed, why?, ret
%d\n", ret);
                break;
            }
        }

        i++;
    }

    if (h)
    {
        now = TVUUTIL_GET_SYS_MS64();
        printf("now %" PRId64 ", finish writing counts %d, would
call LibShmMediaDestroy\n", now, count);
        LibShmMediaDestroy(h); // destory the shared memory.
        h   = NULL;
    }
    return 0;
}




int main()
{
    return WriteSampleCode("test", 10, 10240);
}
```

```
#include "libshm_media.h"
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>
#include <inttypes.h>

#define TVUUTIL_GET_SYS_MS64()
_xxtvuutil_get_sys_ms64()
#define VIDEO_FRAME_SIZE                          10240
#define TVU_LINUX 1

static inline
int64_t _xxtvuutil_get_sys_ms64()
{
#ifdef TVU_WINDOWS
    int64_t tmNow = 0;
    struct _timeb timebuffer;
    _ftime_s(&timebuffer);
    tmNow = timebuffer.time;
    tmNow *= 1000;
    tmNow += timebuffer.millitm;
    return tmNow;
#elif defined(TVU_MINGW)
    int64_t tmNow = 0;
    struct _timeb timebuffer;
    _ftime(&timebuffer);
    tmNow = timebuffer.time;
    tmNow *= 1000;
    tmNow += timebuffer.millitm;
    return tmNow;
#elif defined(TVU_LINUX)
    int64_t tmNow = 0;
    struct timeval  tv;
    gettimeofday(&tv, NULL);
    tmNow = tv.tv_sec * 1000 + tv.tv_usec/1000;
    return tmNow;
#endif
}

static void Sleep(uint32_t ms)
```

```
{
    usleep(ms*1000);
}

static bool g_exit = false;

static int ReadSampleCode(const char *name)
{
    unsigned int                i       = 0;
    libshm_media_handle_t       h       = NULL;
    libshm_media_head_param_t   ohp     = {0};
    const char                  *shmname= name;
    int                         ret     = -1;
    int                         timeout = 3000;//3000 ms

    h       = LibShmMediaOpen(shmname, NULL, NULL); // open the
shared memory, the callback can be NULL here.

    if (!h) { /* failed if the handle was null */
        printf("open [%s] libshm media handle failed\n",
shmname);
        return -1;
    }

    printf("shm version : %x\n", LibShmMediaGetVersion(h));

    while (!g_exit) {
        libshm_media_item_param_t   ohi = {0};
        libshm_media_item_param_t   *datactx    = NULL;

        ret     = LibShmMediaPollReadData(h, &ohp, &ohi,
timeout); // read out the data with timeout.
        uint32_t r_index = LibShmMediaGetReadIndex(h);
        uint32_t w_index = LibShmMediaGetWriteIndex(h);
        int64_t now = TVUUTIL_GET_SYS_MS64(); // to get the
current system time.

        if (ret < 0)
        {
            printf("poll readable ret %d\n", ret);
            break;
```

```
        }
        else if (ret == 0) {
            /**
             *  timeout, but no data
             *  you can go on to wait, or break
             *  Here, 3 seconds no data, I choose break
             */
            //break;
            usleep(1000);
            continue;
        }

        datactx     = &ohi;
        {
            char vsample[5] = {"null"};
            if (datactx->p_vData)
            {
                strncpy(vsample, (char *)datactx->p_vData, 4);
            }
            char asample[5] = {"null"};
            if (datactx->p_aData)
            {
                strncpy(asample, (char *)datactx->p_aData, 4);
            }
            char exsample[5] = {"null"};
            if (datactx->p_CCData)
            {
                strncpy(exsample, (char *)datactx->p_CCData, 4);
            }
            char subsample[5] = {"null"};
            if (datactx->p_sData)
            {
                strncpy(subsample, (char *)datactx->p_sData, 4);
            }
            printf("now %" PRId64 " readout, rindex %u, windex
%u, head[video fourcc 0x%08x, audio fourcc 0x%08x"
                ",vbr %d, sarw %d, sarh %d, src %dx%d, dst %dx%d,
duration %d, scale %d"
                ", channels %x, depth %d, samplerate %d]"
                ", v[%d, %" PRId64 ", %" PRId64 ", %s ...]\n
a[%d, %" PRId64 ", %" PRId64 ", %s ...]\n "
```

```
                "s[%d, %" PRId64 ", %" PRId64 ", %s]\n ext[%d,
%s], user data[%d, 0x%x, %" PRId64 "]\n"
                , now
                , r_index, w_index
                , ohp.u_videofourcc, ohp.u_audiofourcc
                , ohp.i_vbr, ohp.i_sarw, ohp.i_sarh
                , ohp.i_srcw, ohp.i_srch
                , ohp.i_dstw, ohp.i_dsth
                , ohp.i_duration, ohp.i_scale
                , ohp.i_channels, ohp.i_depth, ohp.i_samplerate
                , datactx->i_vLen
                , datactx->i64_vpts
                , datactx->i64_vpts
                , vsample
                , datactx->i_aLen
                , datactx->i64_apts
                , datactx->i64_adts
                , asample
                , datactx->i_sLen
                , datactx->i64_spts
                , datactx->i64_sdts
                , subsample
                , datactx->i_CCLen
                , exsample
                , datactx->i_userDataLen
                , datactx->i_userDataType
                , datactx->i64_userDataCT
            );

            if (ohi.p_userData && ohi.i_userDataLen>0 &&
ohi.i_userDataType == LIBSHM_MEDIA_TYPE_TVU_EXTEND_DATA_V2)
            {
                libshmmedia_extend_data_info_t myExt;
                {
                    memset(&myExt, 0, sizeof(myExt));
                }
                LibShmMeidaParseExtendData(&myExt,
ohi.p_userData, ohi.i_userDataLen, ohi.i_userDataType); // parse
out the extension data.
            }
        }
```

```
    }

    if (h)
    {
        LibShmMediaDestroy(h); // destroyed the shared memory.
        h   = NULL;
    }
    return 0;
}


int main()
{
    return ReadSampleCode("test");
}
```