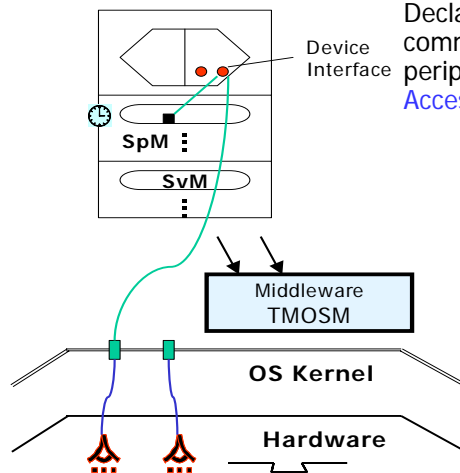


**EECS223:**  
**Real-Time Computer Systems**  
**Programming I/O Activities in TMO -- Pt. 1**

Apr-05 | 1



**Basic Styles of Writing Commands to Peripherals into TMOs**



Declare **wrappers** of the APIs for commanding and controlling peripherals in the **EAC (Environment Access Capability)** section.

- **Wrappers** =: Objects in which methods are included for invoking the kernel APIs for passing commands to peripherals.
- ODSSs to be used during a TMO method execution are generally locked at the time of initiating the method execution.

=> Harmonious sharing of wrappers by concurrently running TMO method executions

Figure 4. Peripheral commands from a TMO via a device interface in the ODS

Apr-05 | 2



## Basic Styles of Writing Reactive Control of Peripherals into TMOs

### RC1: Polling by a dedicated SpM

- [Dedicate an SpM](#) for monitoring the state of the peripheral and record on occurrence of a special condition into an ODSS (An atomically accessible group of data members).
- Check the state of a peripheral through I/O APIs provided by the (commercial) kernel part of the TMO execution engine.

If main application functionalities are in SpMs and SvMs, the above dedicated SpM can be viewed as a [slave function](#).

=> The information (or signal) flow from the peripheral to main application functionalities is asynchronous.

- Good enough in a large number of applications, but [the amount of execution engine resources consumed by the dedicated SpM](#) can become a factor of concern in certain demanding RT applications.

Apr-05	3
--------	---

UCI  
DREAM Lab



## Example Programs

For example programs using

- Basic Styles of Writing [Commands](#) to Peripherals into TMOs and
- Basic Styles of Writing [Reactive Control](#) of Peripherals into TMOs
  - Polling by a dedicated SpM,

See [Appendix A](#) (Audio Capture and Play) and [Appendix B](#) (Video Play)

Apr-05	4
--------	---

UCI  
DREAM Lab



## Why TMOSLprintf ?

- **TMOSLprintf** () is a wrapper function of `printf` (), which enables TMOSM (TMO Support Middleware) to handle printing commands from (various places in ) a TMO in a reliable manner.
- Calling `printf` () directly is not safe because:
  - `printf` () involves sending output data to one global buffer from which the data is printed out.
  - The `printf` () of one thread can be **preempted** by that of another thread.

Apr-05	5
--------	---



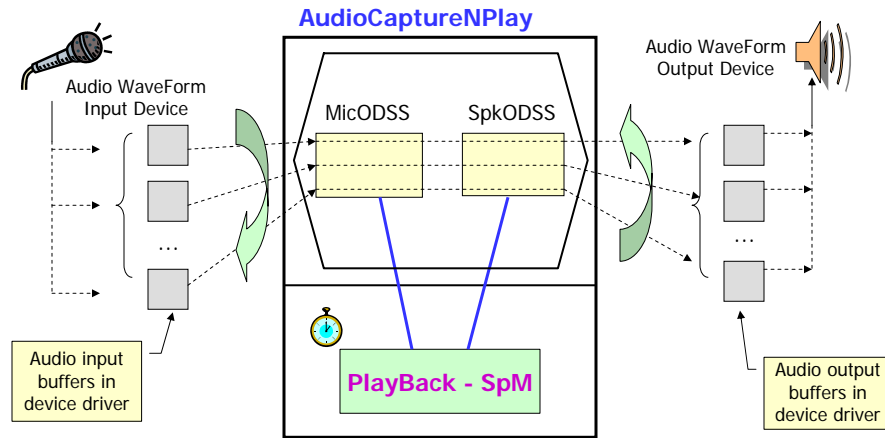
## Appendix A

**Audio Capture & Play TMO**  
**SpM Performing I/O Activities**

Apr-05	6
--------	---



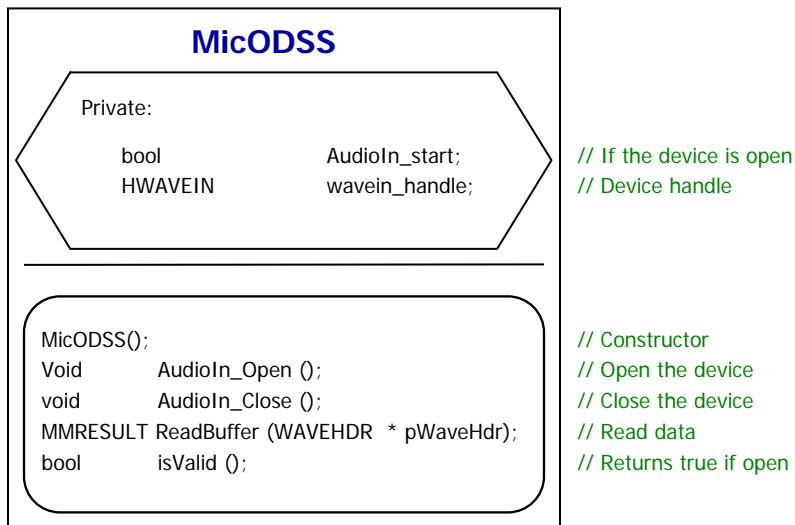
## Audio Capture & Play TMO



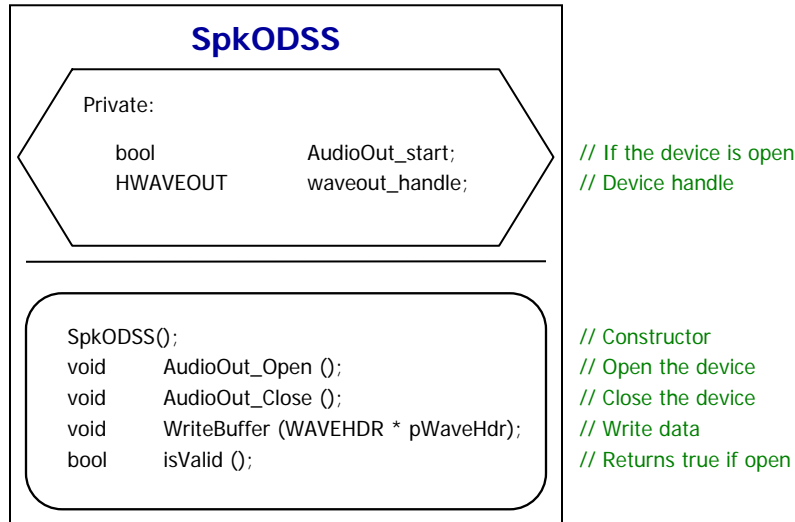
**Aim** - To implement a TMO program that contains a single TMO to read data from the waveform-audio input device and play it on the audio output device.



## MicODSS



## SpkODSS



// If the device is open  
// Device handle

// Constructor  
// Open the device  
// Close the device  
// Write data  
// Returns true if open

Apr-05 9

UCI  
DREAM Lab 

## Audio Capture & Play TMO program

**Aim** - To implement a TMO program that contains a single TMO to read data from the waveform-audio input device and play it on the audio output device.

**An SpM polls the audio input device to collect the input data and sends the collected data to the audio output device.**

**The constructor** - CAudioCaptureNPlay::CAudioCaptureNPlay

1. Register one SpM - **Playback ()**
2. Open the waveform input device (**MicODSS.AudioIn\_Open ()**) and output device (**SpkODSS.AudioOut\_Open ()**).  
Audio input device and audio output device are wrapped by two ODSSs, **MicODSS** for microphone and **SpkODSS** for speaker.  
It also initializes the input and output waveform headers/buffers.

**The spontaneous function** - **Playback ()**

1. Read data from the audio-input buffers and write the data into the audio-output device buffers.

UCI  
DREAM Lab 

## Audio Capture & Play TMO program

---

### **MicODSS.AudioIn\_Open ()** -

1. Opens the waveform-audio input device for reading input data.
2. Initializes the buffers used for receiving data from the audio input device.

### **SpkODSS.AudioOut\_Open ()** -

1. Opens the waveform-audio output device for writing output data.
2. Initializes the buffers used for receiving data from the audio output device.



## Audio Capture & Play TMO program

---

The audio waveform input device writes data into the WaveForm input headers (buffers).

These buffers are polled constantly by a spontaneous method – [Playback](#).

[Playback](#) then puts the picked data into the Waveform output headers (buffers) and invokes the audio waveform output device to playback the data.



## Audio Capture & Play TMO program

When using WAVEFORMATEX structure for the format of waveform-audio data, we need to specify the sampling rate of 8.0kHz, 11.025kHz, 22.05kHz, or 44.1kHz.

We pick 8.0kHz for this sample program and the audio block size of 200 bytes. Then, the polling rate becomes

$$\begin{aligned} \text{Polling period} &= (1 / (8000 / \text{WAVEIN\_AUDIO\_BLOCK\_SIZE})) \\ &= 25 \text{ msec} \end{aligned}$$

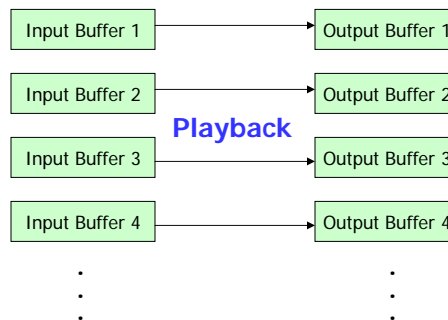


## Eliminating callback

We have eliminated the need for callbacks.

We implement an one-to-one mapping of the input buffers onto the output buffers, i.e., we define an equal number of input and output buffers and the spontaneous method [Playback](#) copies the data in the input buffer to the corresponding output buffer.

[Playback](#) then invokes the waveOutWrite API to have the output waveform device play the data.



## Audio Capture & Play TMO program

```
/******  
*  
* System : Implement a TMO program that contains a single TMO  
* to read data from the waveform-audio input device  
* and play it on the audio output device.  
*  
* Filename : AudioCaptureNPlay.cpp  
*  
* Date : Jan. 29, 2005.  
*  
* Description : This program is used to collect audio data from  
* the audio input buffer and send the data buffer to  
* the audio output device. We use an SpM to regularly  
* poll the input device for data. The SpM then sends  
* the collected data to the output device.  
*  
*  
*****
```



## AudioCaptureNPlay.h

```
// AudioCaptureNPlay.h  
  
#pragma once // Specifies that the file will be included (opened)  
// only once by the compiler in a build.  
  
#include "TMOSL.h" // TMOSL header  
  
#pragma comment (lib, "TMOSL") // TMOSL library will be linked to this program.  
#pragma comment (lib, "Winmm") // Multimedia library for wave in/out will be  
// linked to this program.  
  
using namespace TMO;
```



## AudioCaptureNPlay.h

```
// ODSS for Audio Input
class MicODSS : public ODSSBaseClass <MicODSS>
{
public:
    MicODSS()    { AudioIn_start = false; }
    void        AudioIn_Open ();
    void        AudioIn_Close ();
    MMRESULT    ReadBuffer (WAVEHDR * pWaveHdr);
    bool        isValid () { return AudioIn_start; }

private:
    bool        AudioIn_start;           // flag for microphone start
    HWAVEIN     wavein_handle;          // the handle of WaveIn device
};
```



## AudioCaptureNPlay.h

```
// ODSS for Audio Output
class SpkODSS : public ODSSBaseClass <SpkODSS>
{
public:
    SpkODSS ()  { AudioOut_start = false; }
    void        AudioOut_Open ();
    void        AudioOut_Close ();
    void        WriteBuffer (WAVEHDR * pWaveHdr);
    bool        isValid () { return AudioOut_start; }

private:
    bool        AudioOut_start;         // flag for speaker start
    HWAVEOUT    waveout_handle;        // the handle of WaveOut device
};
```



## AudioCaptureNPlay.h

```
// CAudioCaptureNPlay TMO class definition
class CAudioCaptureNPlay : public CTMOBase
{
public:
    CAudioCaptureNPlay (TCHAR * tmo_name, tms start_time);
    virtual ~CAudioCaptureNPlay ();
private:
    MicODSS      MODSS;          // ODSS for AudioIn (microphone)
    SpkODSS      SODSS;          // ODSS for AudioOut (speaker)

    void  PlayBack ();           // SpM
    void  CloseDevice ();        // close audio in and out devices
};
```



## AudioCaptureNPlay.h

```
// Constants
/* WAVEFORMATEX.nSamplesPerSec;
   Sample rate, in samples per second (hertz), at which each channel should be played or
   recorded. If wFormatTag is WAVE_FORMAT_PCM, then common values for
   nSamplesPerSec are 8.0 kHz, 11.025 kHz, 22.05 kHz, and 44.1 kHz. */

#define N_SAMPLES_PER_SEC      8000
#define WAVEIN_AUDIO_BLOCK_SIZE 200 // WaveIn audio block / buffer size
#define IN_BUFFER              10  // number of WaveIn buffers

#define WAVEOUT_AUDIO_BLOCK_SIZE 200 // WaveOut audio block / buffer size
#define OUT_BUFFER              10  // number of WaveOut buffers

// frequency & period for Playback SpM
#define SAMPLING_FREQUENCY (N_SAMPLES_PER_SEC / WAVEIN_AUDIO_BLOCK_SIZE)
    // unit = Hz
#define SAMPLING_PERIOD (1000 * 1000 / SAMPLING_FREQUENCY)
    // unit = microsec, 25 msec with 8kHz sampling rate & 200 byte buffer size
```



## AudioCaptureNPlay.cpp

```
// AudioCaptureNPlay.cpp

#include "AudioCaptureNPlay.h"
#include "mmsystem.h"           // requires "winmm.lib" (cf. AudioCaptureNPlay.h)

WAVEHDR  wavein_wave_hdr[IN_BUFFER];    // headers for each WaveIn buffer
WAVEHDR  waveout_wave_hdr[OUT_BUFFER];  // headers for each WaveOut buffer

int wavein_open_result;    // status flag on WaveIn open operation
int waveout_open_result;  // status flag on WaveOut open operation
```



## AudioCaptureNPlay.cpp (cont.)

```
/*
 * AudioIn_Open opens the waveform-audio input device for reading input data.
 * It also initializes the buffers used for receiving data from the audio
 * input device.
 */
void MicODSS::AudioIn_Open ()
{
    int          i;
    WAVEFORMATEX wave_format;

    // settings for the WaveIn device
    memset ( (void*) &wave_format, 0, sizeof (wave_format));
    wave_format.wFormatTag      = WAVE_FORMAT_PCM;
    wave_format.nChannels       = 1;
    wave_format.nSamplesPerSec  = N_SAMPLES_PER_SEC;
    wave_format.nAvgBytesPerSec = N_SAMPLES_PER_SEC;
    wave_format.nBlockAlign     = 1;
    wave_format.wBitsPerSample  = 8;
    wave_format.cbSize          = 0;
```



## AudioCaptureNPlay.cpp (cont.)

```

/* Open the waveform-audio input device.*/
wavein_open_result = waveInOpen ( (HWAVEIN *) & wavein_handle,
                                  WAVE_MAPPER,
                                  (WAVEFORMATEX *) & wave_format,
                                  (DWORD) 0,
                                  (DWORD) 0,
                                  CALLBACK_NULL);

/* If device open operation fails, print error message.*/
if (wavein_open_result != MMSYSERR_NOERROR)
{
    wavein_handle = NULL;
    TMOslprintf (_T("WaveIn device open failed.\n"));
    return ;
}

```



## AudioCaptureNPlay.cpp (cont.)

```

/* Initialize the waveform-audio input buffer fields.*/
for (i = 0; i < IN_BUFFER; i++)
{
    wavein_wave_hdr[i].lpData = (char *)
        malloc (WAVEIN_AUDIO_BLOCK_SIZE);
    wavein_wave_hdr[i].dwBufferLength =
        WAVEIN_AUDIO_BLOCK_SIZE;
    wavein_wave_hdr[i].dwUser = 1;
    wavein_wave_hdr[i].dwFlags = 0;

    /* Prepare a waveform-audio input.*/
    waveInPrepareHeader (wavein_handle, &wavein_wave_hdr[i],
                        sizeof (WAVEHDR) );
}

/* Send the input buffer to the input device.*/
waveInAddBuffer (wavein_handle, & wavein_wave_hdr[0], sizeof (WAVEHDR));

```



## AudioCaptureNPlay.cpp (cont.)

```

/* Starts input on the given waveform-audio input device.*/
MMRESULT result = waveInStart (wavein_handle);

/* Check for the return code.*/
if (result != MMSYSERR_NOERROR) {
    TMOSLprintf (_T("waveInStart failed.\n"));
    return;
}

AudioIn_start = true;
return;
} // MicODSS::AudioIn_Open ()

```



## AudioCaptureNPlay.cpp (cont.)

```

// Close AudioIn device.
void MicODSS::AudioIn_Close ()
{
    // Stop input on the given waveform-audio input device.
    if (waveInReset (wavein_handle) != MMSYSERR_NOERROR)
    {
        TMOSLprintf (_T("Could not stop input on the given waveform-audio input
device and reset the current position to zero.\n"));
    }
    else
    {
        TMOSLprintf (_T("Stopped input on the given waveform-audio input device
and reset the current position to zero.\n"));
    }
}

```



## AudioCaptureNPlay.cpp (cont.)

```
// Close the given waveform-audio input device.
if (waveInClose (wavein_handle) != MMSYSERR_NOERROR)
{
    TMOSLprintf (_T("Could not close the given waveform-audio input
device.\n"));
}
else
{
    TMOSLprintf (_T("Closed the given waveform-audio input device.\n"));
    AudioIn_start = false;
}
}
```



## AudioCaptureNPlay.cpp (cont.)

```
// Read data from AudioIn device.
MMRESULT MicODSS::ReadBuffer (WAVEHDR *pWaveHdr)
{
    MMRESULT    result;

    /* Send the input buffer to the input device. */
    result = waveInAddBuffer (wavein_handle, pWaveHdr, sizeof(WAVEHDR));
    return result;
}
```



## AudioCaptureNPlay.cpp (cont.)

```
// AudioOut_Open opens the waveform-audio output device for playback.
// It also initializes the buffers used for sending data to the audio output device.
void SpkODSS::AudioOut_Open ()
{
    int          i;
    int          result;
    WAVEFORMATEX waveout_wave_format;

    // settings for waveform-audio output device
    memset ((void*) & waveout_wave_format, 0, sizeof (waveout_wave_format));
    waveout_wave_format.wFormatTag      = WAVE_FORMAT_PCM;
    waveout_wave_format.nChannels       = 1;
    waveout_wave_format.nSamplesPerSec  = N_SAMPLES_PER_SEC;
    waveout_wave_format.nAvgBytesPerSec = N_SAMPLES_PER_SEC;
    waveout_wave_format.nBlockAlign     = 1;
    waveout_wave_format.wBitsPerSample  = 8;
    waveout_wave_format.cbSize          = 0;
```



## AudioCaptureNPlay.cpp (cont.)

```
/* Open the waveform-audio output device.*/
result = waveOutOpen ( (HWAVEOUT *) & waveout_handle,
                      WAVE_MAPPER,
                      (WAVEFORMATEX *) & waveout_wave_format,
                      (DWORD) 0,
                      (DWORD) 0,
                      CALLBACK_NULL );

/* If device open operation fails, print error message.*/
if (result != MMSYSERR_NOERROR) {
    waveout_handle = NULL;
    TMOSLprintf (_T("WaveOut device open failed.\n"));
    return ;
}
```



## AudioCaptureNPlay.cpp (cont.)

```

/* Initialize the waveform-audio output buffer fields.*/
for (i = 0; i < OUT_BUFFER; i++)
{
    waveout_wave_hdr[i].lpData = (char*) malloc(WAVEOUT_AUDIO_BLOCK_SIZE);
    waveout_wave_hdr[i].dwBufferLength = WAVEOUT_AUDIO_BLOCK_SIZE;
    waveout_wave_hdr[i].dwUser      = 0;
    waveout_wave_hdr[i].dwFlags     = 0L;

    /* Prepare a waveform-audio data block for playback.*/
    waveOutPrepareHeader (waveout_handle, &waveout_wave_hdr[i],
                          sizeof (WAVEHDR));
}

AudioOut_start = true;
return;
} // SpkODSS::AudioOut_Open ()

```



## AudioCaptureNPlay.cpp (cont.)

```

// Close AudioOut device.
void SpkODSS::AudioOut_Close ()
{
    /* Stop playback on the given waveform-audio output device.
    if (waveOutReset(waveout_handle) != MMSYSERR_NOERROR)
    {
        TMOSLprintf (_T("Could not stop playback on the given waveform-audio
        output device and resets the current position to zero.\n"));
    }
    else
    {
        TMOSLprintf (_T("Stopped playback on the given waveform-audio output
        device and resets the current position to zero.\n"));
    }
}

```



## AudioCaptureNPlay.cpp (cont.)

```

// Close the given waveform-audio output device.
if (waveOutClose (waveout_handle) != MMSYSERR_NOERROR)
{
    TMOSLprintf (_T("Could not close the waveform-audio output device.\n"));
}
else
{
    TMOSLprintf (_T("Closed the waveform-audio output device.\n"));
    AudioOut_start = false;
}
}

// Write data to AudioOut device.
void SpkODSS::WriteBuffer (WAVEHDR *pWaveHdr)
{
    waveOutWrite (waveout_handle, pWaveHdr, sizeof(WAVEHDR));
}

```



## AudioCaptureNPlay.cpp (cont.)

```

/*
 * The spontaneous method Playback reads data from the audio-input buffers and
 * writes the data into the audio-output device buffers.
 */
void CAudioCaptureNPlay::Playback ()
{
    MMRESULT    result;
    WAVEHDR     *pWaveHdr;

    /* If both the waveform-audio output and input devices have been initialized.*/
    if ( MODSS.isValid () && SODSS.isValid () )
    {
        /* for every input buffer...*/
        for (int i = 0; i < INPUT_BUFFER; i++)
        {
            pWaveHdr = (WAVEHDR*) & wavein_wave_hdr[i];
            pWaveHdr->dwBufferLength = WAVEOUT_AUDIO_BLOCK_SIZE;

```



## AudioCaptureNPlay.cpp (cont.)

```

/* Send the input buffer to the input device.*/
result = MODSS.ReadBuffer (pWaveHdr);
char waveout_data[WAVEOUT_AUDIO_BLOCK_SIZE];

if (result == MMSYSERR_NOERROR) { // data was received
    memcpy (waveout_data, pWaveHdr->lpData, pWaveHdr->dwBytesRecorded);
    memcpy (waveout_wave_hdr[i].lpData, waveout_data,
            pWaveHdr->dwBytesRecorded);
    waveout_wave_hdr[i].dwUser = 1;
    waveout_wave_hdr[i].dwBufferLength = WAVEOUT_AUDIO_BLOCK_SIZE;
    waveout_wave_hdr[i].dwBytesRecorded = pWaveHdr->dwBytesRecorded;

    /* Send the data block to the waveform-audio output device.*/
    SODSS.WriteBuffer (&waveout_wave_hdr[i]);
} /* End of - If return code = MMSYSERR_NOERROR*/
} /* End of - For every input buffer.*/
} /* End of - if both devices opened.*/
} // Playback ()

```



## AudioCaptureNPlay.cpp (cont.)

```

/*
 * TMO constructor -
 * Register SpM and TMO.
 * Initializes the Audio Input and Audio output devices.
 */
CAudioCaptureNPlay::CAudioCaptureNPlay (TCHAR * tmo_name, tms start_time)
{
    MicroSec    from = 2;
                from *= 1000 * 1000;           // 2 sec
    MicroSec    until = 1 * 60 * 60;
                until *= 1000 * 1000;         // 1 hour
    AAC aac (    NULL,
                tm4_DCS_age (from),           // from
                tm4_DCS_age (until),          // until
                SAMPLING_PERIOD,              // every
                0,                             // est
                15 * 1000,                     // 1st
                25 * 1000);                   // by

    SpM_RegistParam    spm_spec;
    spm_spec.build_regist_info_AAC (aac);
}

```



## AudioCaptureNPlay.cpp (cont.)

```
spm_spec.build_regist_info_ODSS (MODSS.GetId(), RW);
spm_spec.build_regist_info_ODSS (SODSS.GetId(), RW);
RegisterSpM ( (PFSpMBody) Playback, & spm_spec);
```

```
TMO_RegistParam      tmo_spec;
_tcscpy (tmo_spec.global_name, tmo_name);
tmo_spec.start_time = start_time;
RegisterTMO (& tmo_spec);
```

```
/* Initialize the AudioOut & AudioIn devices. */
SODSS.AudioOut_Open ();
MODSS.AudioIn_Open ();
```

```
}
```

```
// Destructor
```

```
CAudioCaptureNPlay::~CAudioCaptureNPlay ()
```

```
{
}
```



## AudioCaptureMain.cpp

```
/* The main function */
```

```
int main ()
```

```
{
```

```
    StartTMOengine ();
```

```
    tms start_time = tm4_DCS_age (2 * 1000 * 1000);
```

```
    CAudioCaptureNPlay audioCaptureNPlay (_T("AudioCaptureTMO"), start_time);
```

```
    MainThrSleep ();
```

```
    return 0;
```

```
}
```



**Appendix B**  
**Video Play TMO**  
**SpM Performing I/O Activities**

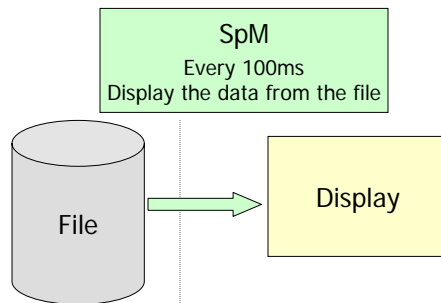
Apr-05 | 39



**WebCamPlayTMO**

This application records a video stream from a web camera into a disk file.

Also the application can display a video stream stored in a file.



Apr-05 | 40



## WebCamPlayTMO.h

---

```
#pragma once
#define WIN32_LEAN_AND_MEAN // Exclude rarely-used stuff from Windows headers
#include <windows.h>

// TMOSL header
#include "TMOSL.h"

// TMOSL library
#pragma comment ( lib, "TMOSM" )

// WinCapDraw
#include "WinCapDraw.h"
#pragma comment ( lib, "wincapdraw" )

using namespace TMO;
```



## WebCamPlayTMO.h

---

```
#define NUMOFBUFFER 2
#define BUFFERSIZE 10
#define EMPTY 0
#define WRITING 1
#define FINISHED 2
#define FULL 3
```



## WebCamPlayTMO.h

```
// ODSS for Video Capture
class VideoCaptureODSS : public ODSSBaseClass <VideoCaptureODSS>
{
public:
    VideoCaptureODSS (HWND hDrawWnd, int x, int y, int width, int height,
        BOOL bVisi) ;
    ~VideoCaptureODSS() {
        delete m_hWinDraw;
        fclose (pFileStream);
    };

    int ReadFrame (char *, size, fSize);

    FILE * pFileStream;
    WinDraw * m_hWinDraw; // WinDraw is in WinCapDraw.DLL.
};
```

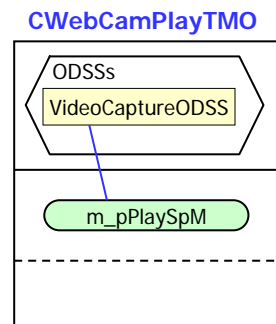


## WebCamPlayTMO.h

```
// class: WebCamPlayTMO
class CWebCamPlayTMO : public CTMOBase
{
private:
    VideoCaptureODSS * pVODSS;

    unsigned int nBytes;
    unsigned char l_szYUVData [CIF_X*CIF_Y*2];
    bool init;
    int nFrameNumber;

    void m_pPlaySpM ();
};
```



## WebCamPlayTMO.h

---

```
public:
    CWebCamPlayTMO (TCHAR *, AAC &, tms &);

    ~CWebCamPlayTMO ()
    {
    };
};
```



## WebCamPlayTMO.cpp

---

```
#include "WebCamPlayTMO.h"

#define WARMUP_DELAY_SECS 8
#define SYSTEM_LIFE_HOURS 24

// Read a frame from the file, pFileStream.
int VideoCaptureODSS::ReadFrame (char * pData, int size, int fSize)
{
    return fread ( pData, size, fSize, pFileStream );
}
```



## WebCamPlayTMO.h

```
// ODSS constructor
VideoCaptureODSS::VideoCaptureODSS (HWND hDrawWnd, int x, int y, int
width, int height, BOOL bVisi)
{
    m_hWinDraw = new WinDraw (hDrawWnd, x, y, width, height, bVisi);
    m_hWinDraw->InitDraw (8);

    pFileStream = fopen ("test2.wcd", "rb"); // read mode,
                                           // byte-formatted file
}

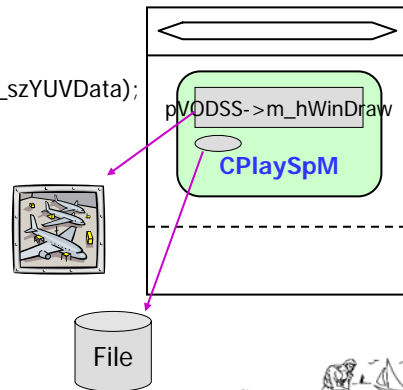
```



## WebCamPlayTMO.cpp

```
void CWebCamPlayTMO::m_pPlaySpM ()
{
    int i = 0;
    i = pVODSS->ReadFrame (L_szYUVData, sizeof (unsigned char), 76032);
    // Frame size is 76032 bytes.
    if (i > 0)
    {
        pVODSS->m_hWinDraw->PicDraw (L_szYUVData);
        ++nFrameNumber;
        TMOSLprintf ("%d", nFrameNumber);
    }
    TMOSLprintf ("\n");
}

```



## WebCamPlayTMO.cpp

```

CWebCamPlayTMO::CWebCamPlayTMO (TCHAR * TMO_name, AAC &
aac_spec, tms & TMO_start_time1)
{
    // Window Creation
    WNDCLASS      wc;

    wc.lpszClassName= "PlayDisplay"; // Pointer to a null-terminated
        // string or is an atom. It specifies the window class name.
    wc.style      = CS_OWNDC | CS_HREDRAW | CS_VREDRAW;
    // Specifies the class style(s).
    wc.lpfnWndProc = (WNDPROC) DefWindowProc;
    // Pointer to the window procedure.
    wc.cbClsExtra  = 0; // Specifies the number of extra bytes to
        // allocate following the window-class structure.

```



## WebCamPlayTMO.cpp

```

wc.cbWndExtra  = 0;
    // Specifies the number of extra bytes to
    // allocate following the window instance.
wc.hInstance   = GetModuleHandle (NULL);
    // Handle to the
    // instance that contains the window procedure for the class.
wc.hIcon       = NULL;
    // Handle to the class icon.
wc.hCursor     = LoadCursor (NULL, IDC_ARROW);
    // Handle to the class cursor.
wc.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
    // Handle to the class background brush.
wc.lpszMenuName = NULL;
    // Pointer to a null-terminated
    // character string that specifies the resource name of the class

```



## WebCamPlayTMO.cpp

```

HWND hWnd = CreateWindow ( wc.lpszClassName,
                          // registered class name
                          "PlayDisplay",
                          // window name
                          WS_OVERLAPPED | WS_THICKFRAME | WS_VISIBLE,
                          // window style
                          400, // horizontal position of window
                          300, // vertical position of window
                          QCIF_X+GetSystemMetrics (SM_CXBORDER)*2,
                          // window width
                          QCIF_Y+GetSystemMetrics (SM_CYCAPTION)
                          + GetSystemMetrics (SM_CYBORDER)*2,
                          // window height
                          NULL, // handle to parent or owner window
                          NULL, // menu handle or child identifier
                          GetModuleHandle (NULL), // handle to application instance
                          NULL // window-creation data
);

```



## WebCamPlayTMO.cpp

```

Sleep (1000);
// Sleep was introduced to give the system enough
// time to create window and etc.
// Instantiate WinDraw - Drawing Object

// Instantiate VODSS.
pVODSS = new VideoCaptureODSS(hWnd, 0, 0, QCIF_X, QCIF_Y, TRUE);
Sleep (1000);
nFrameNumber = 0;

// register SpM
SpM_RegistParam          spm_spec;
spm_spec.build_regist_info_AAC (aac_spec);
RegisterSpM ( (PFSpMBody) m_pPlaySpM, & spm_spec);

```



## WebCamPlayTMO.cpp

```

// register TMO
TMO_RegistParam  tmo_spec;
_tcscpy (tmo_spec.global_name, TMO_name);
tmo_spec.start_time = TMO_start_time1;
RegisterTMO (& tmo_spec);
}

```



## WebCamPlayTMO.cpp

```

void main ()
{
    StartTMOengine ();
    tms TMO_start_time = tm4_DCS_age (7 * 1000 * 1000);
    MicroSec          from = WARMUP_DELAY_SECS;
                    from *= 1000 * 1000;
    MicroSec          until = SYSTEM_LIFE_HOURS * 60 * 60;
                    until *= 1000 * 1000;
    AAC  aac1 ( NULL,
              tm4_DCS_age (from),      // from
              tm4_DCS_age (until),    // until
              100 * 1000,             // every
              0,                      // est
              5 * 1000,               // lst
              50 * 1000);             // by
    CWebCamPlayTMO  webCamPlayTMO (_T("TMO1"), aac1, TMO_start_time);
    MainThrSleep ();
}

```



## backup



## StdAfx.h

---

```
// stdafx.h : include file for standard system include files,  
// or project specific include files that are used frequently, but  
// are changed infrequently  
  
#if !defined  
    (AFX_STDAFX_H__B1409B15_75F8_11D3_BF6B_00A0CC3FBC6E__INCLUDED_)  
#define AFX_STDAFX_H__B1409B15_75F8_11D3_BF6B_00A0CC3FBC6E__INCLUDED_  
  
#if _MSC_VER > 1000  
#pragma once  
#endif  
                // _MSC_VER > 1000  
  
// Insert your headers here  
#define WIN32_LEAN_AND_MEAN                // Exclude rarely-used stuff from  
    Windows headers  
  
#include <windows.h>  
  
// TMOsl header  
#include <tmosl.h>
```



## StdAfx.h

```
// TMOSL library
#ifdef _DEBUG
#pragma comment ( lib, "tmosl_d" )
#else
#pragma comment ( lib, "tmosl" )
#endif

// TODO: reference additional headers your program requires here

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined
(AFX_STDAFX_H__B1409B15_75F8_11D3_BF6B_00A0CC3FBC6E__INCLUDED_)
```



## Converting an IP address to InetAddr

- API: `unsigned long inet_addr (const TCHAR* cp);`
- Example:  
`unsigned long Addr;`  
`Addr = inet_addr ("128.195.164.141");`  
then,  
`Addr = 0x 8da4c380`

