

EECS123:
**Introduction to
Real-Time Distributed Programming**
Programming I/O Activities in TMO -- Pt. 1

Feb-07	1
--------	---

UCI
DREAM Lab



Introduction

- High-level real-time distributed computing objects are generally written in forms independent of execution platforms
 - However, input and output (I/O) activities involving peripherals are inherently platform-dependent.
- Writing parts of real-time objects for controlling peripherals should be done **in forms compatible with the adopted real-time object programming styles.**
- A desirable goal :
 - To facilitate both **commanding** and **reactive control** of peripherals **in general forms while enabling relatively easy analysis of the timing behavior.**

Feb-07	2
--------	---

UCI
DREAM Lab



Basic Styles of Writing Commands to Peripherals into TMOs

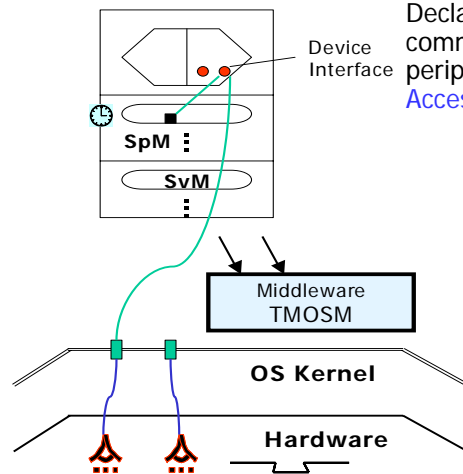


Figure 4. Peripheral commands from a TMO via a device interface in the ODS

Feb-07 3

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

UCI
DREAM Lab



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.

Feb-07 4

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)

Feb-07	5
--------	---

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.

Feb-07	6
--------	---

UCI
DREAM Lab



Appendix A

Audio Capture & Play TMO

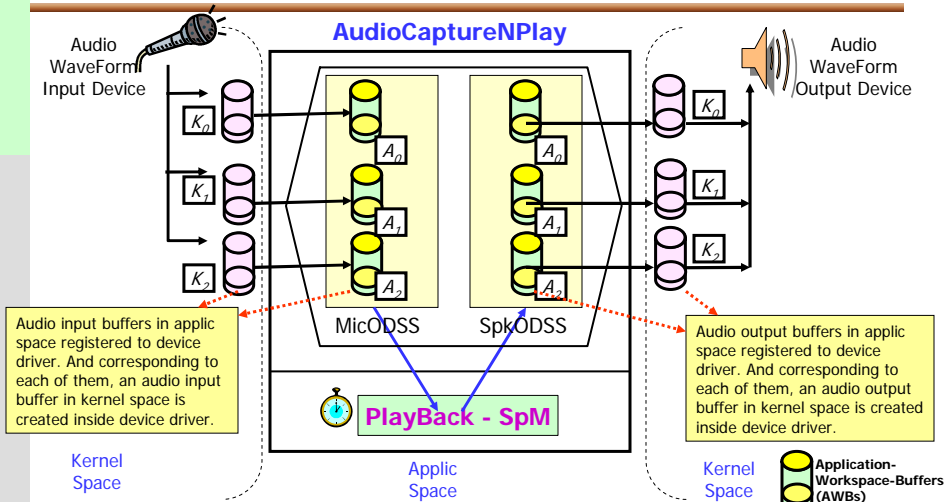
SpM Performing I/O Activities

Feb-07 7

UCI
DREAM Lab



Audio Capture & Play TMO

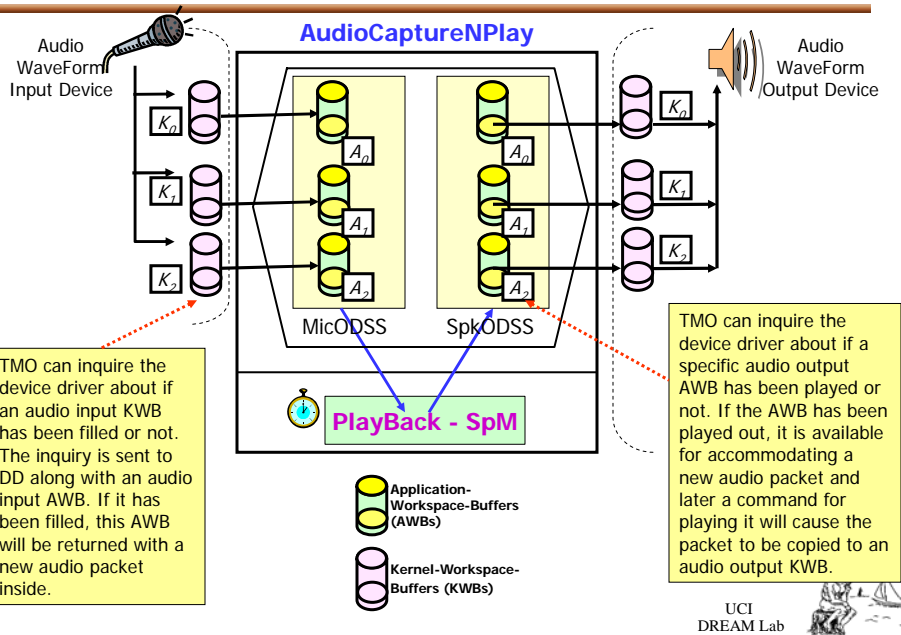


Aim - To implement a TMO program that contains a single TMO to read data from the waveform-audio input device and play it on the audio output device.

UCI
DREAM Lab



Operating Audio Devices with Status Polling



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.OpenMic ()`) and output device (`SpkODSS.OpenSpk ()`).
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 device and write the data into the audio-output device.

Audio Capture & Play TMO program

MicODSS.OpenMic 0 -

1. Opens the audio input device for obtaining audio packets.
2. Initializes and registers audio input AWBs used for receiving data from the audio input device.

SpkODSS.OpenSpk 0 -

1. Opens the audio output device for playing audio packets.
2. Initializes and registers audio output AWBs used for sending data to the audio output device.

UCI
DREAM Lab



Audio Capture & Play TMO program

The audio waveform input device writes audio packets into the audio input KWBs.

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

[Playback](#) retrieves audio packets from audio input KWBs and saves them into audio input AWBs through calling the microphone interface, [Read](#) (int buffer_number), which in turn calls Win32 API, waveInAddBuffer().

[Playback](#) then puts the audio packets into the audio output AWBs and invokes the audio waveform output device to playback the data.

UCI
DREAM Lab



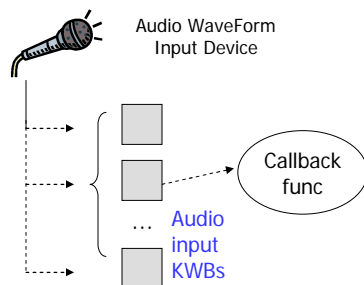
Audio Capture & Play TMO program

- 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.
- Also, we need to specify the number of channels: 1 or 2.
- And, we need to specify the number of bits per sample: 8bits or 16 bits.
- We choose the sample rate as 44.1kHz, the number of channels as 2 and the number of bits per sample as 16 for this sample program. Also, we set the period of the PlayBack SpM as 25ms. Then, the size of each audio packet is calculated as follows.

$$\begin{aligned}
 \text{WAVEIN_AUDIO_BLOCK_SIZE} &= ((\text{CHANNEL_NUM} * \text{SAMPLE_SIZE} \\
 &\quad * \text{SAMPLE_RATE} * \text{SPM_PERIOD})/1000) \\
 &= 2*2*44100*25/1000 = 4410 \text{ bytes}
 \end{aligned}$$

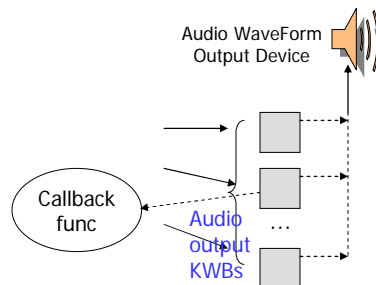


Operating Audio Devices in Callback Mode



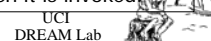
Once the device driver has filled an audio input KWB, it invokes a callback function registered by the application.

Inside the callback function, an audio input AWB is provided to device driver to copy the new audio packet from the audio input KWB to the audio input AWB.



Once the device driver has played an audio packet in a specific audio output KWB, it invokes a callback function registered by the application.

A parameter of the callback function carries a pointer to an audio output AWB which sent an audio packet to the audio output KWB whose content has just been played when it is invoked.



Current Playback-SpM Uses Polling

- Since callback functions run outside the TMO domain, they are not adopted in the current Audio Capture & Play implementation.
- **Playback** then invokes calling the speaker interface, **Play** (char * waveout_dataPtr), which in turn calls Win32 API, waveOutWrite(), to have the output waveform device play the data.

UCI
DREAM Lab



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 : Feb. 16, 2006.  
*  
* Description : This program is used to collect audio data from  
* the audio input device and send the data 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.  
*  
*  
*****
```

UCI
DREAM Lab



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;
```

UCI
DREAM Lab



AudioCaptureNPlay.h

```
// ODSS for Audio Input
class Mic_Wrapper_Class : public ODSSBaseClass <Mic_Wrapper_Class>
{
public:
    Mic_Wrapper_Class () {};
    ~Mic_Wrapper_Class () {};
    MMRESULT waveInOpen( LPHWAVEIN phwi, UINT_PTR uDeviceID,
                        LPWAVEFORMATEX pwf, DWORD_PTR dwCallback,
                        DWORD_PTR dwCallbackInstance, DWORD fdwOpen
                        );
    MMRESULT waveInPrepareHeader(HWAVEIN hwi, LPWAVEHDR pwh,
                                  UINT cbwh);
    MMRESULT waveInStart( HWAVEIN hwi );

    MMRESULT waveInAddBuffer( HWAVEIN hwi, LPWAVEHDR pwh,
                              UINT cbwh );
```

Vendor-
Supplied
Interfaces

UCI
DREAM Lab



AudioCaptureNPlay.h

```

void          OpenMic ();          // Initialize the WaveIn device.
BOOL         IsMicStart () { return AudioIn_start;}
MMRESULT     Read (int buffer_number);
char*        GetBufferData (int buffer_number) { return AWB[buffer_number];}

private:
char AWB[IN_BUFFER][WAVEIN_AUDIO_BLOCK_SIZE];
WAVEFORMATEX wave_format; // WaveIn Device Setting Parameter
HWAVEIN      wavein_handle; // The handle of WaveIn device.
WAVEHDR      wavein_wave_hdr[IN_BUFFER];
              // The headers for each WaveIn buffer.
BOOL         AudioIn_start; // Flag for microphone start.
};

```

UCI
DREAM Lab



AudioCaptureNPlay.h

```

// ODSS for Audio Output
class SpkODSS : public ODSSBaseClass <SpkODSS>
{
public:
    Spk_Wrapper_Class () {};
    ~Spk_Wrapper_Class () {};
    MMRESULT waveOutOpen(LPHWAVEOUT phwo, UINT_PTR uDeviceID,
        LPWAVEFORMATEX pwf,   DWORD_PTR dwCallback,
        DWORD_PTR dwCallbackInstance, DWORD fdwOpen );
    MMRESULT waveOutPrepareHeader(
        HWAVEOUT hwo, LPWAVEHDR pwh, UINT cbwh );
    MMRESULT waveOutWrite(
        HWAVEOUT hwo, LPWAVEHDR pwh, UINT cbwh);
};

```

Vendor-
Supplied
Interfaces

UCI
DREAM Lab



AudioCaptureNPlay.h

```

void      OpenSpk ();      // Initialize the WaveOut device.
BOOL     IsSpkStart () { return AudioOut_start;}
void     Play (char * waveout_dataPtr); // Method for Playing Audio Packets
private:
char     AWB[OUT_BUFFER][WAVEIN_AUDIO_BLOCK_SIZE];
WAVEFORMATEX  wave_format; // WaveOut Device Setting Parameter
HWAVEOUT  waveout_handle; // The handle of WaveOut device.
WAVEHDR   waveout_wave_hdr[OUT_BUFFER];
// The headers for each WaveOut buffer.
BOOL     AudioOut_start; // Flag for speaker start.
};

```

UCI
DREAM Lab



AudioCaptureNPlay.h

```

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

    void Playback (); // SpM
};

```

UCI
DREAM Lab



AudioCaptureNPlay.h

```
// Constants
#define SPM_PERIOD 25 // The period of PlayBack SpM
#define SPM_DEADLINE 20 // The deadline of PlayBack Spm

#define SAMPLE_RATE 44100 // Sample rate
#define SAMPLE_SIZE 2 // The number of bits per sample
#define CHANNEL_NUM 2 // The number of channels
#define WAVEIN_AUDIO_BLOCK_SIZE ((CHANNEL_NUM * SAMPLE_SIZE *
    SAMPLE_RATE * SPM_PERIOD)/1000) // WaveIn audio block/buffer size.
#define IN_BUFFER 3 // The number of WaveIn buffers.

#define WAVEOUT_AUDIO_BLOCK_SIZE ((CHANNEL_NUM * SAMPLE_SIZE *
    SAMPLE_RATE * SPM_PERIOD)/1000) // WaveOut audio block/buffer size.
#define OUT_BUFFER 3 // The number of WaveOut buffers.
```

UCI
DREAM Lab



AudioCaptureNPlay.cpp

```
// AudioCaptureNPlay.cpp

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

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```
MMRESULT Mic_Wrapper_Class::waveInOpen(  
    LPHWAVEIN phwi,  UINT_PTR  uDeviceID,  
    LPWAVEFORMATEX pwx,  DWORD_PTR  dwCallback,  
    DWORD_PTR  dwCallbackInstance,  DWORD  fdwOpen  
    ) {  
    return waveInOpen(  
        phwi, uDeviceID, pwx, dwCallback, dwCallbackInstance, fdwOpen);  
    }  
MMRESULT Mic_Wrapper_Class::waveInPrepareHeader(  
    HWAVEIN hwi,  LPWAVEHDR pwh,  
    UINT cbwh  
    ) {  
    return waveInPrepareHeader (hwi,  pwh,  cbwh);  
    }
```

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```
MMRESULT Mic_Wrapper_Class::waveInStart(  
    HWAVEIN hwi  
    )  
    {  
    return waveInStart(hwi);  
    }  
MMRESULT Mic_Wrapper_Class::waveInAddBuffer(  
    HWAVEIN hwi,  LPWAVEHDR pwh,  UINT cbwh  
    )  
    {  
    return waveInAddBuffer(hwi,  pwh,  cbwh);  
    }
```

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```

/*
 * OpenMic 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 Mic_Wrapper_Class::OpenMic ()
{
    // settings for the WaveIn device
    memset ( (void*) &wave_format, 0, sizeof (wave_format));
    wave_format.wFormatTag      = WAVE_FORMAT_PCM;
    wave_format.nChannels      = CHANNEL_NUM;
    wave_format.nSamplesPerSec = SAMPLE_RATE;
    wave_format.nAvgBytesPerSec = SAMPLE_RATE;
    wave_format.nBlockAlign    = 1;
    wave_format.wBitsPerSample = 8 * SAMPLE_SIZE;
    wave_format.cbSize         = 0;

```

UCI
DREAM Lab



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 ;
}

```

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```

/* Initialize the header fields for audio input AWBs.*/
for (i = 0; i < IN_BUFFER; i++)
{
    wavein_wave_hdr[i].lpData          = (char *) AWB[i];
    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) );
}

```

UCI
DREAM Lab



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.Wn"));
    return;
}

AudiIn_start = true;
return;
} // MicODSS::OpenMic ()

```

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```

// Read data from AudioIn device.
MMRESULT Mic_Wrapper_Class::Read (int buffer_number)
{
    MMRESULT    result;
    WAVEHDR     *pWaveHdr;

    pWaveHdr    =      (WAVEHDR*) &wavein_wave_hdr[buffer_number];
    pWaveHdr->dwBufferLength    =    WAVEOUT_AUDIO_BLOCK_SIZE;

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

```

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```

MMRESULT Spk_Wrapper_Class::waveOutOpen(
    LPHWAVEOUT phwo,    UINT_PTR    uDeviceID,
    LPWAVEFORMATEX pwf,    DWORD_PTR    dwCallback,
    DWORD_PTR    dwCallbackInstance,    DWORD    fdwOpen
)
{
    return waveOutOpen( phwo,    uDeviceID,    pwf,
        dwCallback,    dwCallbackInstance,    fdwOpen);
}

MMRESULT Spk_Wrapper_Class::waveOutPrepareHeader(
    HWAVEOUT hwo,    LPWAVEHDR pwh,    UINT cbwh
)
{
    return waveOutPrepareHeader(hwo, pwh, cbwh);
}

```

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```
MMRESULT Spk_Wrapper_Class::waveOutWrite(
    HWAVEOUT hwo,
    LPWAVEHDR pwh,
    UINT cbwh
)
{
    return waveOutWrite(
        hwo,
        pwh,
        cbwh
    );
}
```

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```
// OpenSpk opens the waveform-audio output device for playback.
// It also initializes the buffers used for sending data to the audio output device.
void Spk_Wrapper_Class::OpenSpk ()
```

```
{
    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 = CHANNEL_NUM;
waveout_wave_format.nSamplesPerSec = SAMPLE_RATE;
waveout_wave_format.nAvgBytesPerSec = SAMPLE_RATE;
waveout_wave_format.nBlockAlign = 1;
waveout_wave_format.wBitsPerSample = 8 * SAMPLE_SIZE;
waveout_wave_format.cbSize = 0;
```

UCI
DREAM Lab



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.Wn"));
    return ;
}

```

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```

/* Initialize the header fields of audio output AWBs.*/
for (i = 0; i < OUT_BUFFER; i++)
{
    waveout_wave_hdr[i].lpData = (char*) AWB[i];
    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::OpenSpk ()

```

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```

// Send data to AudioOut device.
void Spk_Wrapper_Class::Play (char *waveout_dataPtr)
{
    int    buf;

    for    (buf = 0; buf < OUT_BUFFER;    buf++)
    {
        DWORD flag = waveout_wave_hdr[buf].dwFlags;
        // Check if an audio output AWB can be used
        if(flag == (WHDR_DONE | WHDR_PREPARED) || flag == WHDR_PREPARED)
        {
            memcpy ( waveout_wave_hdr[buf].lpData,
                    (char *) (waveout_dataPtr),
                    WAVEOUT_AUDIO_BLOCK_SIZE);

            waveout_wave_hdr[buf].dwBufferLength = WAVEOUT_AUDIO_BLOCK_SIZE;
            // Send this AWB to device driver
            int res = waveOutWrite (waveout_handle, &waveout_wave_hdr[buf], sizeof (WAVEHDR));
            return;
        }
    }
    return;
}

```

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```

/*
 * The spontaneous method Playback reads data from the audio-input device 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 (MicODSS.IsMicStart () && SpkODSS.IsSpkStart ())
    {

```

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```

for (int i = 0; i < IN_BUFFER; i++)
{
    /* Send an audio input AWB to the input device.*/
    result = MicODSS.Read (i);

    /* return code = MMSYSERR_NOERROR => data was received.*/
    if (result == MMSYSERR_NOERROR)
    {
        /* Send the audio packet to the waveform-audio output device.
        SpkODSS.Play (MicODSS.GetBufferData (i));
        break;
    }
}
} // Playback ()

```

UCI
DREAM Lab



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
                  SPM_PERIOD * 1000,   // every
                  0,                    // est
                  15 * 1000,           // 1st
                  SPM_DEADLINE * 1000); // by

    SpM_RegistParam      spm_spec;
    spm_spec.build_regist_info_AAC (aac);

```

UCI
DREAM Lab



AudioCaptureNPlay.cpp (cont.)

```
spm_spec.build_regist_info_ODSS (MicODSS.GetId(), RW);
spm_spec.build_regist_info_ODSS (SpkODSS.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. */
SpkODSS.OpenSpk ();
MicODSS.OpenMic ();
```

```
}
```

```
// Destructor
```

```
CAudioCaptureNPlay::~CAudioCaptureNPlay ()
{
}
```

UCI
DREAM Lab



AudioCaptureMain.cpp

```
#include "AudioCaptureNPlay.h"
```

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

```
int main ()
```

```
{
```

```
    StartTMOengine ();
```

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

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

```
    MainThrSleep ();
```

```
    return 0;
```

```
}
```

UCI
DREAM Lab



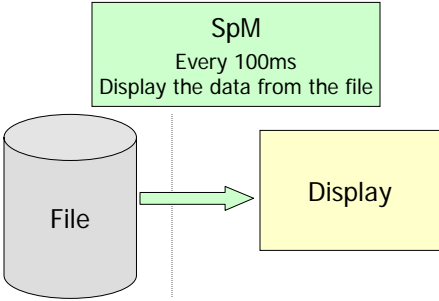
Appendix B
Video Play TMO
SpM Performing I/O Activities

Feb-07 43



WebCamPlayTMO

- A video stream from a web camera has already been recorded into a disk file.
- This application can display a video stream stored in a file.



Feb-07 44



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, "TMOSL" )

// 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 File Device Interfaces in Win32
class FileDeviceODSS : public ODSSBaseClass <FileDeviceODSS>
{
public:
    FILE    * pFileStream;

    //Constructor & Destructor
    FileDeviceODSS () {}
    ~FileDeviceODSS() {}
};
```



WebCamPlayTMO.h

```
// Visual C++ Run-Time Library
FILE * fopen ( const char *filename, const char *mode );
int fclose ( FILE *stream );
size_t fread ( void *buffer, size_t size, size_t count, FILE *stream );
size_t fwrite ( const void *buffer, size_t size, size_t count, FILE
*stream );
};
```

Vendor-Supplied
Interfaces



WebCamPlayTMO.h

```
// ODSS for Display Device Interface
class VideoDisplayODSS : public ODSSBaseClass <VideoDisplayODSS>
{
private:
    WinDraw * m_hWinDraw; // WinDraw is a class defined in wincapdraw.lib.
    HWND     hWnd; // HWND is defined in windef.h
public:
    VideoDisplayODSS () {}
    ~VideoDisplayODSS () {}
};
```

UCI
DREAM Lab



WebCamPlayTMO.h

```
void PicDraw (unsigned char * pstream);
HWND CreateWindowX ( LPCTSTR lpClassName,
                    LPCTSTR lpWindowName,
                    DWORD dwStyle,
                    int x,
                    int y,
                    int nWidth,
                    int nHeight,
                    HWND hWndParent,
                    HMENU hMenu,
                    HINSTANCE hInstance,
                    void * lpParam);
BOOL InitDraw (int nDitherType) ;

BOOL openWindow (int x, int y, int width, int height, BOOL bVisi);
void closeWindow ();
};
```

Vendor-Supplied
Interfaces

UCI
DREAM Lab



WebCamPlayTMO.h

```

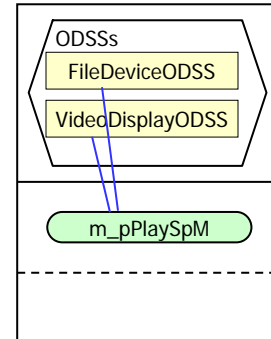
// class: WebCamPlayTMO
class CWebCamPlayTMO : public CTMOBase
{
private:
    FileDeviceODSS      * pFDODSS;
    VideoDisplayODSS   * pVDPODSS;

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

    void m_pPlaySpM ();

```

CWebCamPlayTMO



UCI
DREAM Lab



WebCamPlayTMO.h

```

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

    ~CWebCamPlayTMO () ;
};

```

UCI
DREAM Lab



WebCamPlayTMO.cpp

```
#include "stdafx.h"
#include "VideoCamPlayTMO.h"

#define WARMUP_DELAY_SECS 8
#define SYSTEM_LIFE_HOURS 24
```

UCI
DREAM Lab



WebCamPlayTMO.cpp

```
FILE * FileDeviceODSS::fopen (const char *filename, const char *mode)
{ return ::fopen (filename, mode); }

int FileDeviceODSS::fclose ( FILE *stream )
{ return ::fclose (stream); }

size_t FileDeviceODSS::fread ( void *buffer, size_t size, size_t count, FILE
*stream)
{ return ::fread ( buffer, size, count, stream); }

size_t FileDeviceODSS::fwrite
( const void *buffer, size_t size, size_t count, FILE *stream) { return ::fwrite
(buffer, size, count, stream); }
```

UCI
DREAM Lab



WebCamPlayTMO.cpp

```

VideoDisplayODSS::PicDraw (unsigned char * pstream) {
    return m_hWinDraw->PicDraw (pstream);
}

HWND VideoDisplayODSS::CreateWindowX ( LPCTSTR lpClassName,
LPCTSTR lpWindowName, DWORD dwStyle, int x, int y, int nWidth, int
nHeight, HWND hWndParent, HMENU hMenu, HINSTANCE hInstance, void
* lpParam ) {
    return ::CreateWindow ( lpClassName, lpWindowName, dwStyle,
x, y, nWidth, nHeight, hWndParent, hMenu, hInstance, lpParam);
}

BOOL VideoDisplayODSS::InitDraw (int nDitherType) {
    return m_hWinDraw->InitDraw (nDitherType);
}

```

UCI
DREAM Lab



WebCamPlayTMO.cpp

```

VideoDisplayODSS::openWindow (int x, int y, int width, int height, BOOL bVisi)
{
    // Window Creation
    WNDCLASS wc;

    wc.lpszClassName = _T("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.
}

```

UCI
DREAM Lab



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

```

UCI
DREAM Lab



WebCamPlayTMO.cpp

```

HWND hWnd = CreateWindowX ( wc.lpszClassName, // registered class name
    _T("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
);

```

UCI
DREAM Lab



WebCamPlayTMO.cpp

```

m_hWinDraw = new WinDraw (hWnd, x, y, width, height, bVisi);
if (m_hWinDraw == NULL)
    return FALSE;

    InitDraw (8);
    return TRUE;
}

void VideoDisplayODSS::closeWindow ()
{
    delete m_hWinDraw;
}

```

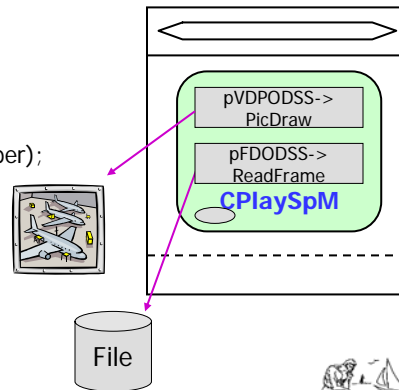


WebCamPlayTMO.cpp

```

void CWebCamPlayTMO::m_pPlaySpM ()
{
    int i = 0;
    i = pFDODSS->odss_fread (l_szYUVData, sizeof (unsigned char), 38016,
pFDODSS->pFileStream); // Frame size is 38016 bytes.
    if (i > 0)
    {
        pVDPODSS->PicDraw (l_szYUVData);
        ++nFrameNumber;
        TMOSLprintf (_T("%d"),nFrameNumber);
    }
    TMOSLprintf (_T("Wn"));
}
}

```



WebCamPlayTMO.cpp

```

CWebCamPlayTMO::CWebCamPlayTMO (TCHAR * TMO_name, AAC &
aac_spec, tms & TMO_start_time1)
{
    // Instantiate ODSS.
    pFDODSS = new FileDeviceODSS();
    pFDODSS->pFileStream = pFDODSS->fopen("test2.wcd", "rb");
    pVDPODSS = new VideoDisplayODSS();
    pVDPODSS->openWindow(0, 0, QCIF_X, QCIF_Y, TRUE);
    nFrameNumber = 0;

    // register SpM
    SpM_RegistParam    spm_spec;
    spm_spec.build_regist_info_AAC    (aac_spec);
    spm_spec.build_regist_info_ODSS(pVFLODSS->GetId(), RO);
    spm_spec.build_regist_info_ODSS(pVDPODSS->GetId(), RW);
    RegisterSpM ( (PFSpMBody) m_pPlaySpM, & spm_spec);

```

UCI
DREAM Lab



WebCamPlayTMO.cpp

```

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

CWebCamPlayTMO::~CWebCamPlayTMO()
{
    pFDODSS->fclose(pFDODSS->pFileStream);
    pVDPODSS->closeWindow();
}

```

UCI
DREAM Lab



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
             125 * 1000,             // every
             0,                      // est
             20 * 1000,              // lst
             100 * 1000);            // by
    CWebCamPlayTMO webCamPlayTMO (_T("TMO1"), aac1, TMO_start_time);
    MainThrSleep ();
}
```

