

**EECS 123:**

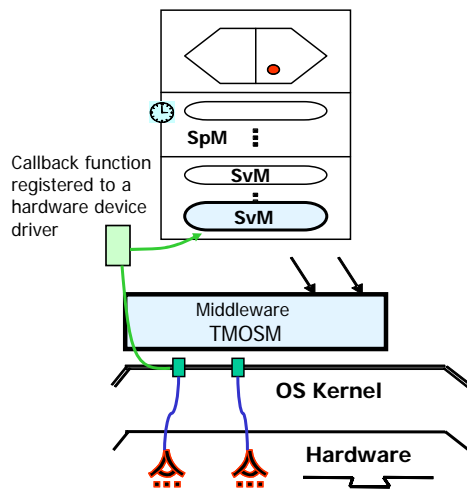
**Introduction to  
Non-TMO-to-TMO OnewaySR  
Programming**

**Programming I/O Activities in TMO -- Pt. 2**

Feb-07 1



**Basic styles of writing reactive control –  
RC2: SvM invoked Indirectly by a Device Driver**



Feb-07 2

- To write an SvM which is **invoked upon occurrence of a specified signal from the underlying OS kernel.**
- Not broadly applicable.
  - The device driver must be built to use a callback function.
  - Often, ISRs (and DPC target functions) of peripherals do not seem to provide necessary hooks.
  - So, often necessary to modify an ISR to schedule an invocation of (or a DPC for) a callback function which will in turn invoke the SvM.



## OneWaySR from Non-TMO threads to TMO

- An API for calling from non-TMO functions (e.g., callback functions) for SvMs in TMOs.
- Both non-TMO functions and TMOs to be called should be on the same node with an IP address.
- TMOsL provides a class **CGate\_4\_NonTMO** which has a member function serving as an API for calling an SvM.

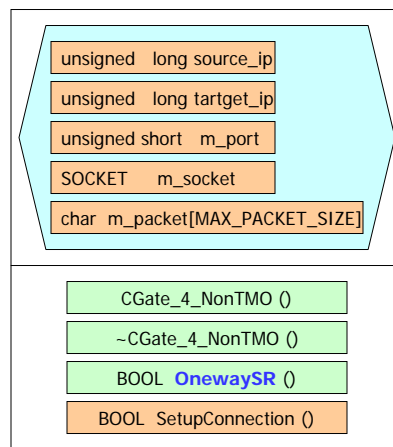
Feb-07 3



## CGate\_4\_NonTMO

```
class CGate_4_NonTMO
{
public:
    CGate_4_NonTMO ();
    ~CGate_4_NonTMO ();
    BOOL OnewaySR (
        TCHAR * target_tmo_name,
        TCHAR * target_svm_name,
        void* ParamPtr, int ParamSize);
private:
    BOOL SetupConnection (unsigned short port);
    unsigned short m_port;
    unsigned long source_ip;
    unsigned long target_ip;
    SOCKET m_socket;
    char m_packet [MAX_PACKET_SIZE];
};
```

- Used to instantiate a gate for the TMO world in the local node



Feb-07 4



## CGate\_4\_NonTMO in TMOSL

- Used to instantiate a gate for the TMO world in the local node
- Object method

```
BOOL OneWaySR (TCHAR * target_tmo_name, TCHAR *
               target_svm_name, void* ParamPtr, int ParamSize);
```

- `target_tmo_name`  
[in] name of a target TMO
  - `target_svm_name`  
[in] name of a target SvM to be triggered
  - `ParamPtr`  
[in] pointer to a parameter-message to be sent
  - `ParamSize`  
[in] size of the parameter-message which ParamPtr is pointing to
- \* Note that there is `no ORT` parameter. -- This is different from the case of `OneWaySR ()` used inside the TMO world.

Return value

- `TRUE` is returned if the function succeeds.
- Otherwise, `FALSE` is returned.

Feb-07

5

UCI  
DREAM Lab

## Example 1

### TestNonTMOonewaySR.h

```
typedef struct _ParamStruct
{
    TCHAR content[200];
    int size;
} ParamStruct;

class TMO1: public CTMOBase
{
public:
    TMO1 (TCHAR * TMO_name, TCHAR *
          RMMC_name, tms TMO_start_time);
private:
    //The SvM to call
    int SvM1 (ParamStruct *);
};
```

### TestNonTMOonewaySR.cpp

```
TMO1::TMO1 (TCHAR * TMO_name, TCHAR *
            SvM_name, tms TMO_start_time)
{
    SvM_RegistParam svm_regist_param;
    _tcscopy (svm_regist_param.name, SvM_name);
    // Register svm along with the registration
    // parameter
    BOOL result = RegisterSvM ( (PFSvMBody)
                               SvM1, & svm_regist_param);

    // fill out tmo registration parameters
    TMO_RegistParam tmo_regist_param;
    _tcscopy (tmo_regist_param.global_name,
             TMO_name);
    tmo_regist_param.start_time =
        TMO_start_time;

    // register this TMO to the TMOSM
    result = RegisterTMO (& tmo_regist_param);
}
```

Feb-07

6

UCI  
DREAM Lab

## Example 1 (cont.)

### TestNonTMOOnewaySR.cpp (cont')

```
int TMO1::SvM1 (ParamStruct * param)
{
    //Print out the message
    TMO_Sprintf (_T("Received a message::
    %s\n"), param->content);
    return 1;
}
```

- Since a non-TMO program cannot utilize TMO services, the sending timestamp cannot be generated.

==> Inside the SvM function body, a call for `GetSRTimestamp ()` must not appear.

```
//Non-TMO thread
void ThreadRoutine (void)
{
    //Create a communication gate for NonTMO
    CGate_4_NonTMO gate;

    //Prepare the data to be sent
    TCHAR buf [200];
    _tcscpy (buf, _T("This is a message to be sent from
    NonTMO thread to TMO method.Wn"));
    ParamStruct param;
    param.size = sizeof (ParamStruct);
    _tcscpy (param.content, buf);

    //Send out the Data
    gate.OnewaySR (_T("TMO1"), _T("SvM1"),
    (void*) & param, sizeof (ParamStruct));
}
```

Feb-07

7

UCI  
DREAM Lab

## Example 1 (cont.)

```
void main ()
{
    StartTMOengine ();
    tms TMO_start_time1 = tm4_DCS_age (3*1000*1000);
    TMO1 tmo1 (_T("TMO1"), _T("SvM1"), TMO_start_time1);
    //Create a non-TMO thread
    HANDLE nonTMOThread = CreateThread
        (NULL, FALSE,
        (LPTHREAD_START_ROUTINE) ThreadRoutine,
        NULL, FALSE, NULL);
    MainThrSleep ();
}
```

Feb-07

8

UCI  
DREAM Lab

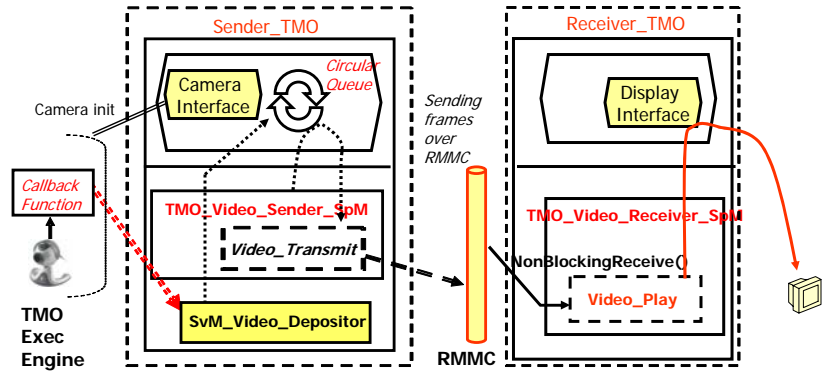
## Example 2

### Tele-Video

Feb-07 9



## The Structure of Tele-Video



Feb-07 10



## Tele-Video Sender



### main.cpp

---

```
#include "StdAfx.h"
#include "TMO_Video_Send.h"

int main ()
{
    // Start TMO support Middleware
    StartTMOengine (NULL);
    tms    start_time = tm4_DCS_age ( 5*1000*1000);

    // Instantiate the TMO_Video_Sender TMO object
    TMO_Video_Send_Class* pTMO_Video_Send = new
    TMO_Video_Send_Class (_T("TMO_Video_Send"), start_time);

    // Main thread goes to sleep and TMOSM takes the control
    MainThrSleep ();
    return 0;
}
```



## VideoCam.h

---

```
// VideoCam.h

#ifndef __VIDEO_CAM_H__
#define __VIDEO_CAM_H__

#include "StdAfx.h"
#include "Wincap.h"

using namespace TMO;
```



## VideoCam.h

---

```
// The class is the device wrapper for video camera
class VideoCamODSSClass : public ODSSBaseClass <VideoCamODSSClass>
{
public:
    VideoCamODSSClass ();
    ~VideoCamODSSClass () {};

    // Initialize the video camera with "VideoCamInitParam"
    int          VideoCamInit (VideoInitParamStruct * VideoCamInitParam);
    // Check if the video camera has been successfully started
    BOOL        IsVideoCamStarted () { return m_VideoCamStart; }
    // Open the video camera
    int          OpenVideoCam ();
```



## VideoCam.h

```
private:
    WinCap* m_hWinCap; // Pointer to the video camera object (class definition and
                       // implementation are omitted)
    VideoInitStruct * m_pVideoCamInitStruct; // Pointer to the
                                             // VideoCamInitStruct

    int      m_x;
    int      m_y;
    int      m_width;
    int      m_height;
    BOOL     m_bVisibility;
    HWND     m_hWnd;

    HWND     CreateVideoWindow ();

    bool     m_VideoCamStart; // Flag for the successful start of video camera
};
#endif
```

Data members and function for the video capturing window

UCI  
DREAM Lab



## VideoInitStruct.h

```
#ifndef __VIDEO_INIT_PARAM_H__
#define __VIDEO_INIT_PARAM_H__

typedef struct _VideoInitStruct
{
    int  x; // the initial horizontal position of the video capturing window
    int  y; // the initial vertical position of the video capturing window
    int  width; // the width of the video capturing window
    int  height; // the height of the video capturing window
    int  FrameRatePerSecond; // the capture frame rate per second
    int  FrameDepth; // the bit length of each pixel
    void * VideoCamStreamCallback; // Pointer to the video callback function
};
#endif
```

UCI  
DREAM Lab



## VideoInitParam.h

```

_VideoInitParamStruct ():
    x (0),
    y (0),
    width (DEFAULT_FRAME_WIDTH),
    height (DEFAULT_FRAME_HEIGHT),
    FrameRatePerSecond (DEFAULT_FRAME_RATE_PER_SECOND),
    FrameDepth (DEFAULT_FRAME_DEPTH),
    VideoCamStreamCallback (NULL) {}

    virtual ~_VideoInitParamStruct () {}
} VideoInitParamStruct;
#endif

```

UCI  
DREAM Lab



## VideoFrame.h

```

#ifndef VIDEO_FRAME_h
#define VIDEO_FRMAE_h

#include "StdAfx.h"
#include "constants.h"

using namespace TMO;
typedef struct _SVideoFrame
{
    unsigned int    nBytes; // the video frame size
    int             nFrameID; // the video frame ID
    int             CodecFmtSize; // the size of the codec format
    unsigned char   CodecFmt [MAX_CODEC_FORMAT_SIZE]; // the codec format
    unsigned char   Data; // Pointer to the video frame
} SVideoFrame;
#endif

```

UCI  
DREAM Lab



## VCM.h

---

```

#ifndef VCM_H
#define VCM_H

#include <windows.h>
#include <Vfw.h>

#define  MJPG  mmioFOURCC ('M','J','P','G')
           // Tells VCM what MJPG represents.

```

UCI  
DREAM Lab



## VCM.h (cont.)

---

```

// The class for the wrapper of codec
class VCM_Codec_Class : public ODSSBaseClass < VCM_Codec_Class >
{
public:
    VCM_Codec_Class ();
    ~VCM_Codec_Class ();

    // Encode a video frame
    int  Encode (void * yuvStream, void* MJPGStream);
    // Decode a video frame
    int  Decode (void * yuvStream, void* MJPGStream);
    // Get the codec format
    void* GetCodecFmt ();
    // Get the size of the codec format
    int  GetCodecFmtSize ();
    // Set the codec format
    void SetCodecFmt (void* pCodecFmt, int CodecFmtSize);

```

UCI  
DREAM Lab



## VCM.h (cont.)

```

private:
    HIC          hicc; // Handle of the Image Compression Manager
                  // for MJPEG Decompression
    HIC          hicc; // Handle of the Image Compression Manager
                  // for MJPEG Compression
    LPBITMAPINFOHEADER pRawFmt; // Pointer to the capture's
                  // raw format description
    LPBITMAPINFOHEADER pMJPEGFmt; // Pointer to the
                  // MJPEG format description
    LONG          IFmtLength; // Length of the MJPEG format description
    PVOID         pRawBuffer; // Pointer to the raw data
    PVOID         pMJPEGBuffer; // Pointer to the MJPEG compressed data
    int           Width; // The width of raw image
    int           Height; // The height of raw image
}
#endif

```

UCI  
DREAM Lab



## Wincap\_buffer.h

```

#ifndef _WincapBuffer_
#define _WincapBuffer_

#include "TMOSL.h"
#include "constants.h"

using namespace TMO;

// Auxiliary data structure for the circular buffer
typedef struct _Buffer
{
    char *pBuffer;
    int size;
} Buffer;

```

UCI  
DREAM Lab



## Wincap\_buffer.h

```

// The class defines the circular buffer
class WinCap_Video_Class : public ODSSBaseClass <WinCap_Video_Class>
{
public:
    WinCap_Video_Class () :
        VideoBufferHead (0),
        VideoBufferTail (0),
        LastAction (2) {};
    ~WinCap_Video_Class () {};

    // Write a video frame into the buffer
    bool      WriteVideoFrame (char *, int );
    // Get a video frame from the buffer
    int       GetOneFrame (char *, int*);

```

UCI  
DREAM Lab



## Wincap\_buffer.h

```

private:
    // The buffer used to save video frames
    Buffer      pVideoFrame [VIDEO_BUFFER_NUMBER];
    // The index pointing to the header of the buffer
    unsigned int  VideoBufferHead;
    // The index pointing to the tail of the buffer
    unsigned int  VideoBufferTail;
    // The latest operation on this buffer ( 0:Write, 1:Read, 2: Initial value )
    unsigned int  LastAction;

    // Local functions
    int          CopyFrame (char*, int *);
    bool         IsVideoBufferEmpty ();
    unsigned int  GetVideoBufferHead ();
    unsigned int  GetVideoBufferTail ();
};

```

UCI  
DREAM Lab



## RMMC.h

```
#ifndef RMMC_H
#define RMMC_H

#include "StdAfx.h"
#include "constants.h"

class RMMC_Class : public RMMCgateBaseClass
{
public:
    RMMC_Class (TCHAR* RMMC_name)
    {
        build_regist_info_EM (MAX_MESSAGE_SIZE*sizeof(char));
        RegisterRMMCgate (RMMC_name);
    };
};
#endif // RMMC_H
```



## TMO\_Video\_Send.h

```
#ifndef TMO_VIDEO_SENDER_H
#define TMO_VIDEO_SENDER_H

#include "StdAfx.h"
#include "VideoFrame.h"
#include "RMMC.h"
#include "VideoCam.h"
#include "Wincap_buffer.h"

#include "VCM\VCM.h"

using namespace TMO;
```



## TMO\_Video\_Send.h

```

class TMO_Video_Send_Class: public CTMOBase
{
private:

    int          TMO_Video_Sender_SpM (); // Function body of
                                           // TMO_Video_Sender_SpM

    int          SvM_Video_Depositor (PVOID); // Function body of
                                           // SvM_Video_Depositor

    VideoCamODSSClass * m_pVideoCam; // Wrapper for video camera
                                           // (Implementation is omitted)
    VCM_Codec_Class * pVCM_codec; // Pointer to the codec ODSS (Class
                                           // definition & implementation are omitted) VCM = Video-Compression Mgr
    WinCap_Video_Class VideoBuffer; // Circular buffer ODSS for video frames
                                           // (class definition & implementation are omitted)
    RMMC_Class      RMMC_g; // RMMC gate for sending out video frames

```

UCI  
DREAM Lab



## TMO\_Video\_Send.h

```

// Local functions
void Register_TMO_Video_Sender_SpM (); // Register SpM to TMOSM
void Register_SvM_Video_Depositor (); // Register SvM to TMOSM
void CodecInit (); // Initialize the codec

public:
    TMO_Video_Send_Class (TCHAR *, tms&);

};
#endif

```

UCI  
DREAM Lab



## TMO\_Video\_Send.cpp (cont.)

```

#include "NonTMO2TMO.h"

#include "TMO_Video_Send.h"
#include "constants.h"

using namespace TMO;

CGate_4_NonTMO nonTMOGate; // A NonTMO2TMO gate for
                           // SvM_Video_Depositor
int nVideoFrameID = -1; // Current Video Frame ID - A global variable

```



## TMO\_Video\_Send.cpp (cont.)

```

// Video frame callback function
LRESULT CALLBACK WinCapVideoStreamCallback (HWND hWnd,
LPVIDEOHDR lpVHdr)
{
    int size = 0;
    size = sizeof (SVideoFrame) + lpVHdr->dwBytesUsed;
    char * buf = new char[size];
    SVideoFrame * pVideoFrame = (SVideoFrame *) buf;
    pVideoFrame->nBytes = lpVHdr->dwBytesUsed;
    memcpy ( (void*) &pVideoFrame->Data, (char *) lpVHdr->lpData,
            lpVHdr->dwBytesUsed);
    nVideoFrameID++;
    pVideoFrame->nFrameID = nVideoFrameID;
}

```

Compose a video frame



## TMO\_Video\_Send.cpp (cont.)

---

```
// Invoke the SvM_Video_Depositor by calling OnewaySR
nonTMOGate.OnewaySR (_T("TMO_Video_Send"), _T("VideoData_SvM"),
                    (void *) pVideoFrame, sizeof (void*));

return (TRUE);
} // end: CapWinCapVideoStreamCallback ()

////////////////////////////////////////////////////////////////
```



## TMO\_Video\_Send.cpp (cont.)

---

```
////////////////////////////////////////////////////////////////

void TMO_Video_Send_Class::Register_TMO_Video_Sender_SpM ()
{
    SpM_iteration = 0;

    SpM_RegistParam   TMO_Video_SpM_spec;
```



## TMO\_Video\_Send.cpp (cont.)

```

//specify time window for SpM
MicroSec from = (MicroSec) WARMUP_DELAY_SECS * 1000 * 1000;
MicroSec until = (MicroSec) SYSTEM_LIFE_HOURS * 60 * 60 * 1000 * 1000;
MicroSec every = (MicroSec) ENCODE_SPM_PERIOD * 1000;
MicroSec est = 0;
MicroSec lst = est + LASTEST_SPM_START_TIME * 1000;
MicroSec by = ENCODE_SPM_DEADLINE * 1000;
AAC aac1 (
    NULL, // null for candidate aac label
    tm4_DCS_age ((MicroSec) WARMUP_DELAY_SECS * 1000 * 1000) ,
    tm4_DCS_age ((MicroSec) SYSTEM_LIFE_HOURS * 60 * 60 * 1000 *
        1000),
    every, est,
    lst, by
);

```

UCI  
DREAM Lab



## TMO\_Video\_Send.cpp (cont.)

```

TMO_Video_SpM_spec.build_regist_info_AAC (aac1);
// Register codec, RMMC gate, video circular buffer
// and the wrapper of video camera as ODSSs
TMO_Video_SpM_spec.build_regist_info_ODSS (
    VideoBuffer.GetId(),RO);
TMO_Video_SpM_spec.build_regist_info_ODSS (
    m_pVideoCam->GetId (), RW);
TMO_Video_SpM_spec.build_regist_info_ODSS (
    RMMC_g.GetId (), RW);
TMO_Video_SpM_spec.build_regist_info_ODSS (
    pVCM_codec->GetId (), RW);
// register SpM
if (RegisterSpM ((PFSpMBody) (&TMO_Video_Send_Class::
    TMO_Video_Sender_SpM), &TMO_Video_SpM_spec) == FAIL)
    TMOSLprintf (
        _T("Fail to register TMO_Video_Sender_SpM object \n"));
}

```

UCI  
DREAM Lab



## TMO\_Video\_Send.cpp (cont.)

```
int TMO_Video_Send_Class::TMO_Video_Sender_SpM ()
{
    unsigned int      nBytes;
    int nTempFrameID = -1;

    unsigned char FrameBuf [sizeof (SVideoFrame) + FRAME_WIDTH *
    FRAME_HEIGHT * FRAME_DEPTH / 8];
        // Uncompressed video frame buffer
    unsigned char CompressedFraemBuf [FRAME_WIDTH*FRAME_HEIGHT*2];
        // Compressed video frame buffer
    unsigned char VideoMessage [MAX_MESSAGE_SIZE];
        // Buffer for Video message
```



## TMO\_Video\_Send.cpp (cont.)

```
if (m_pVideoCam && m_pVideoCam->IsVideoCamStarted ()) {
    SVideoFrame * pTempVideoFrame;
    int rtn = VideoBuffer.GetOneFrame (
        (char*) FrameBuf, (int*) & nBytes); // Get one frame from the circular
        // buffer (Implementation omitted)
    if (rtn) {
        pTempVideoFrame = (SVideoFrame *) FrameBuf;
        nTempFrameID = pTempVideoFrame->nFrameID;
    }
}
```



## TMO\_Video\_Send.cpp (cont.)

```

if (nTempFrameID != -1)
{
    if (pVCM_codec) {
        // Encode a frame (Implementation omitted)
        nBytes = pVCM_codec->Encode (
            & pTempVideoFrame->Data,
            CompressedFraemBuf);
    }

```

Compose a  
video  
message

```

int size = 0;
size = sizeof (SVideoFrame) + nBytes;
char * buf = new char [size];
SVideoFrame * pVideoFrame = (SVideoFrame *) buf;
pVideoFrame->nBytes = nBytes;
memcpy ( (void*) & pVideoFrame->Data,
        CompressedFraemBuf, nBytes);
pVideoFrame->nFrameID = nTempFrameID;

```

UCI  
DREAM Lab



## TMO\_Video\_Send.cpp (cont.)

Compose a  
video  
message

```

pVideoFrame->CodecFmtSize =
    pVCM_codec->GetCodecFmtSize ();
memcpy (pVideoFrame->CodecFmt,
        pVCM_codec->GetCodecFmt (),
        pVideoFrame->CodecFmtSize);
memcpy ( (void *) & VideoMessage,
        (void *) pVideoFrame, size );

```

```

// Send out the video message through the RMMC channel
if (!RMMC_g.Announce ( (void *) & VideoMessage,
    MAX_MESSAGE_SIZE*sizeof (char)) == SUCCESS)
    TMOslprintf (_T("Fail to send %d msgWn"),
        pVideoFrame->nFrameID);
else
    TMOslprintf (_T("Succeed to send %d msgWn"),
        pVideoFrame->nFrameID);
delete pVideoFrame;}}

```

```
return 1;}
```

UCI  
DREAM Lab



## TMO\_Video\_Send.cpp (cont.)

```

// SvM invoked by the call of nonTMO2TMO OnewaySR
int TMO_Video_Send_Class::SvM_Video_Depositor (PVOID param)
{
    SVideoFrame * pCurrentVideoFrame = (SVideoFrame *) (param);

    // Save the video frame into the circular buffer (implementation is omitted)
    VideoBuffer.WriteVideoFrame ( (char*) pCurrentVideoFrame, sizeof
        (SVideoFrame) + pCurrentVideoFrame->nBytes);

    char * buf = (char *) pCurrentVideoFrame;
    delete [] buf;

    return 1;
}

```

UCI  
DREAM Lab



## TMO\_Video\_Send.cpp (cont.)

```

// Register the SvM_Video_Depositor into TMOSM
void TMO_Video_Send_Class::Register_SvM_Video_Depositor ()
{
    SvM_RegistParam svm_regist_param;
    _tcscopy (svm_regist_param.name, _T("VideoData_SvM"));
    svm_regist_param.GETB = VIDEO_DATA_SVM_GETB * 1000;
    svm_regist_param.build_regist_info_ODSS (
        VideoBuffer.GetId (), RW);
    BOOL result = RegisterSvM ((PFSvMBody) &TMO_Video_Send_Class::
        SvM_Video_Depositor, &svm_regist_param);
}

```

UCI  
DREAM Lab



## TMO\_Video\_Send.cpp (cont.)

---

```
void TMO_Video_Send_Class::CodecInit ()
{
    // Initialize Encoder (class definition and implementation are omitted)
    pVCM_codec = new VCM_Codec_Class;
}
```



## TMO\_Video\_Send.cpp (cont.)

---

```
TMO_Video_Send_Class::TMO_Video_Send_Class (TCHAR* TMO_name, tms
& start_time):
    RMMC_g (_T("RMMC_Video_Channel")),
    pVCM_codec (NULL)
{
    // Instantiate the object for the wrapper of video camera (class definition and
    // implementation are omitted)
    m_pVideoCam = new VideoCamODSSClass;
```



## TMO\_Video\_Send.cpp (cont.)

```

// Initialize VideoCam
VideoInitStruct * pVideoCamInitStruct = new VideoInitStruct;

pVideoCamInitStruct->x = 0;
pVideoCamInitStruct->y = 0;
pVideoCamInitStruct->width = FRAME_WIDTH;
pVideoCamInitStruct->height = FRAME_HEIGHT;
pVideoCamInitStruct->FrameRatePerSecond =
    FRAME_RATE_PER_SECOND;
pVideoCamInitStruct->VideoCamStreamCallback = (void*)
    WinCapVideoStreamCallback;
m_pVideoCam->VideoCamInit (pVideoCamInitStruct);

// Initialize codec
CodecInit ();

```



## TMO\_Video\_Send.cpp (cont.)

```

// register SvM_Video_Depositor
Register_SvM_Video_Depositor ();

//register TMO_Video_Sender_SpM
Register_TMO_Video_Sender_SpM ();

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

```



## Tele-Video Receiver



### main.cpp

---

```
#include "StdAfx.h"
#include "TMO_Video_Receive.h"

int main ()
{
    // Start TMO support Middleware
    StartTMOengine (NULL);
    tms    start_time = tm4_DCS_age ( 5*1000*1000);

    // Instantiate the TMO_Video_Receiver TMO object
    TMO_Video_Recv_Class * pTMO_Video_Recv = new
        TMO_Video_Recv_Class (_T ("TMO_Video_Recv"),start_time);

    // Main thread goes to sleep and TMOSM takes the control
    MainThrSleep ();
    return 0;
}
```



## VideoDisplay.h (cont.)

```

#ifndef __VIDEO_DISPLAY_H__
#define __VIDEO_DISPLAY_H__

#include "StdAfx.h"
#include "constants.h"
#include "Windraw.h"

#include "VideoInitParam.h"

using namespace TMO;

```



## VideoDisplay.h (cont.)

```

// The wrapper class of the display device (Implementation is omitted)
class VideoDisplayODSSClass : public ODSSBaseClass
    <VideoDisplayODSSClass>
{
public:
    VideoDisplayODSSClass ();
    ~VideoDisplayODSSClass () {};
    // Check if the display device has been successfully started
    BOOL      IsDisplayStarted () { return m_DisplayStart;}
    // Display a video frame
    void      DisplayFrame (void* FrameBuf);
    // Set the latest received video frame ID
    void      SetLastVideoFrameID (unsigned int FrameID)
                { LastVideoFrameID = FrameID;}
    // Get the frame ID from the latest received video frame
    int       GetLastVideoFrameID () { return LastVideoFrameID;}

```



## VideoDisplay.h (cont.)

```
// Open the display device.
void      OpenDisplay ();
// Initialize the display device with the "VideoInitParam"
int       VideoDisplayInit ( VideoInitParamStruct * VideoInitParam );
```

private:

```
WinDraw *m_hWinDraw; // Pointer to the display device
```

int	m_x;
int	m_y;
int	m_width;
int	m_height;
int	m_depth;
BOOL	m_bVisibility;
HWND	m_hWnd;
HWND	CreateDisplayWindow ();

← Data member and  
function related to  
the display window

UCI  
DREAM Lab



## VideoDisplay.h (cont.)

```
bool m_DisplayStart; // Flag for the successful start of the display device

int  LastVideoFrameID; // Frame ID of the latest received video frame

};
#endif
```

UCI  
DREAM Lab



## TMO\_Video\_Receive.h

```

#ifndef TMO_VIDEO_RECEIVE_H
#define TMO_VIDEO_RECEIVE_H

#include "VideoFrame.h"
#include "VideoDisplay.h"
#include "RMMC.h"

#include "VCMWVCM.h"

using namespace TMO;

```



## TMO\_Video\_Receive.h (cont.)

```

class TMO_Video_Recv_Class : public CTMOBase
{
private:
    int                TMO_Video_Receiver_SpM (); // Function body of
                    // TMO_Video_Receiver_SpM

    RMMC_Class        RMMC_g; // the RMMC gate for retrieving video frames
    VCM_Codec_Class * pVCM_codec; // Pointer to the codec ODSS
    VideoDisplayODSSClass * m_pDisplay; // Pointer to the wrapper of
                    // display device

    // Local functions
    void                CodecInit (); // Initialize the codec
    void                Register_TMO_Video_Receiver_SpM (); // Register SpM
public:
    TMO_Video_Recv_Class (TCHAR *, tms&);
    ~TMO_Video_Recv_Class ();
};
#endif

```



## TMO\_Video\_Receive.cpp

```
#include "TMO_Video_Receive.h"

// Register TMO_Video_Receiver_SpM into TMOSM
void TMO_Video_Recv_Class :: Register_TMO_Video_Receiver_SpM ()
{
    SpM_RegistParam    TMO_Video_Receiver_SpM_spec;

    MicroSec every =    (DECODE_SPM_PERIOD ) * 1000;
    MicroSec est   =    0;
    MicroSec lst   =    est + (LASTEST_SPM_START_TIME ) * 1000;
    MicroSec by    =    (DECODE_SPM_DEADLINE ) * 1000;
    AAC aac1 (
        NULL, // null for candidate aac label
        tm4_DCS_age ( (MicroSec) WARMUP_DELAY_SECS),
        tm4_DCS_age ( (MicroSec) SYSTEM_LIFE_HOURS * 60 * 60 * 1000 *
            1000),
        every, est, lst, by );
```

UCI  
DREAM Lab



## TMO\_Video\_Receive.cpp (cont.)

```
TMO_Video_Receiver_SpM_spec.build_regist_info_AAC (aac1);

// Register codec, RMMC gate,
// and the wrapper of display device as ODSSs
TMO_Video_Receiver_SpM_spec.build_regist_info_ODSS (
    m_pDisplay->GetId (), RW);
TMO_Video_Receiver_SpM_spec.build_regist_info_ODSS (
    RMMC_g.GetId (), RW);
TMO_Video_Receiver_SpM_spec.build_regist_info_ODSS (
    pVCM_codec->GetId (), RW);

// register SpM
if (RegisterSpM ((PFSpMBody) (
    &TMO_Video_Recv_Class::TMO_Video_Receiver_SpM),
    &TMO_Video_Receiver_SpM_spec) == FAIL)
    TMOSLprintf (_T("Fail to register Decode_SpM object Wn"));
}
```

UCI  
DREAM Lab



## TMO\_Video\_Receive.cpp (cont.)

```

int TMO_Video_Recv_Class::TMO_Video_Receiver_SpM ()
{
    // Temporary buffers
    unsigned char  FrameBuf [FRAME_WIDTH * FRAME_HEIGHT *
    FRAME_DEPTH / 8];
    unsigned char  CompressedFrameBuf [FRAME_WIDTH*FRAME_HEIGHT*2];
    unsigned char  VideoMessage [MAX_MESSAGE_SIZE];

    int            nBytes;
    tms            Timestamp;
    if (m_pDisplay && m_pDisplay->IsDisplayStarted ())
    {
        // Get a video frame from the RMMC channel
        while ( RMMC_g.NonBlockingReceive (
            &VideoMessage, &nBytes, Timestamp) == SUCCESS)
        {

```

UCI  
DREAM Lab



## TMO\_Video\_Receive.cpp (cont.)

```

SVideoFrame * pVideoFrame = (SVideoFrame*) & VideoMessage;
if (pVideoFrame->nFrameID > m_pDisplay->GetLastVideoFrameID ())
{
    // Record the frame ID
    m_pDisplay->SetLastVideoFrameID (pVideoFrame->nFrameID);
    memcpy (CompressedFraemBuf,
        (void*) &pVideoFrame->Data, pVideoFrame->nBytes);
    // Set the codec format
    pVCM_codec->SetCodecFmt (pVideoFrame->CodecFmt,
        pVideoFrame->CodecFmtSize);
    // Decode a frame
    pVCM_codec->Decode (FrameBuf, CompressedFrameBuf);
    // Display a frame
    m_pDisplay->DisplayFrame (FrameBuf);
}
}
return 1;
}

```

UCI  
DREAM Lab



## TMO\_Video\_Receive.cpp (cont.)

---

```
// Initialize the codec
void TMO_Video_Recv_Class::CodecInit ()
{
    // Instantiate the codec object (class definition and implementation are omitted)
    pVCM_codec = new VCM_Codec_Class;
}
```



## TMO\_Video\_Receive.cpp (cont.)

---

```
TMO_Video_Recv_Class::TMO_Video_Recv_Class (TCHAR* TMO_name, tms
& start_time):
    RMMC_g (_T("RMMC_Video_Channel")),
    pVCM_codec (NULL)
{
    // Instantiate the object for the wrapper of display device
    m_pDisplay = new VideoDisplayODSSClass;

    VideoInitParamStruct * pVideoInitParam = new VideoInitParamStruct;
```



## TMO\_Video\_Receive.cpp (cont.)

---

```
pVideoInitParam->x = 0;
pVideoInitParam->y = 0;
pVideoInitParam->width = FRAME_WIDTH;
pVideoInitParam->height = FRAME_HEIGHT;
pVideoInitParam->FrameDepth = FRAME_DEPTH;

// Initialize the display device with VideoInitParam
m_pDisplay->VideoDisplayInit (pVideoInitParam);
// Initialize the codec
CodecInit ();

//register TMO_Video_Receiver_SpM
Register_TMO_Video_Receiver_SpM ();
```



## TMO\_Video\_Receive.cpp (cont.)

---

```
TMO_RegistParam TMO_Video_Recv_spec;
_tcscpy (TMO_Video_Recv_spec.global_name, TMO_name);
TMO_Video_Recv_spec.start_time = start_time;
RegisterTMO (&TMO_Video_Recv_spec);
};
```

