
MVR-D2000 (Amber) Development Kit

Programmer's Reference

Version.3.22

MVR-D2000 (Amber) Development Kit Technical inquiries should be directed to Canopus development support:

E-mail: amber_sdk@canopus.co.jp

Canopus Co., Ltd
Main office
1-2-2 Murotani, Nishi-ku
Kobe, Hyogo 651-2241
JAPAN

The logo for Canopus, featuring the word "canopus" in a bold, lowercase, sans-serif font. The letters are a dark green color.

Table Of Contents

TABLE OF CONTENTS	2
PRELIMINARIES.....	6
New Functions.....	6
Included Files	7
Installation	8
Visual C++ Application Development	8
<i>Development Environment</i>	<i>8</i>
<i>Compile.....</i>	<i>9</i>
<i>Link</i>	<i>9</i>
Application Development in other environments.....	9
Sample Programs.....	9
Source Code	10
Questions.....	10
TUTORIAL.....	11
Encoder Application Development	11
<i>Selecting the card.....</i>	<i>11</i>
<i>Launching the encoder.....</i>	<i>12</i>
<i>Encoding to files</i>	<i>12</i>
<i>Memory transfer</i>	<i>12</i>
<i>Overlay window creation</i>	<i>13</i>
<i>Changing position and size of the overlay window</i>	<i>14</i>
<i>Starting the encoder</i>	<i>14</i>
<i>Stopping the Encoder.....</i>	<i>15</i>
<i>Overlay window interruption</i>	<i>15</i>
<i>Closing the encoder.....</i>	<i>15</i>
Decoder Application Development	16
<i>Selecting the card.....</i>	<i>16</i>
<i>Launching the decoder.....</i>	<i>17</i>
<i>Decoding files</i>	<i>17</i>
<i>Memory transfer</i>	<i>17</i>
<i>Overlay window creation</i>	<i>18</i>
<i>Changing position and size of the window</i>	<i>19</i>
<i>Starting the decoder</i>	<i>19</i>
<i>Stopping the decoder</i>	<i>20</i>
<i>Overlay window interruption</i>	<i>20</i>
<i>Closing the decoder.....</i>	<i>20</i>
QUICK REFERENCE	21
Video Encoder Functions.....	21
Video Decoder Functions.....	23
VIDEO ENCODER FUNCTIONS.....	25
ENC_Set_Callback	25
ENC_CB_STATUS	25
ENC_CB_ERROR	26
ENC_CB_VOBU_ENT.....	26
ENC_VOBU_ENT_INFO	27
ENC_Can_Initialize.....	28
ENC_Get_Codec_Config	29

ENC_CONFIG.....	29
ENC_Set_Codec_Number	31
ENC_Initialize	32
ENC_Initialize_Ex.....	33
ENC_BSS_PARAMETER	33
ENC_CB_BSS	33
ENC_Terminate	35
ENC_Get_Media	36
ENC_Set_Media.....	37
ENC_Can_Record	38
ENC_Can_Overlay_Window.....	40
ENC_Create_Overlay_Window	41
ENC_Destroy_Overlay_Window	42
ENC_Move_Overlay_Window	43
ENC_Resize_Overlay_Window	44
ENC_Show_Overlay_Window.....	45
ENC_Get_Overlay_Window.....	46
ENC_Get_Overlay_Rect.....	47
ENC_Set_Overlay_Rect.....	48
ENC_Start_Monitor	49
ENC_Stop_Monitor	50
ENC_Get_Monitor_Status	51
ENC_Get_VideoCD_Mode	52
ENC_Set_VideoCD_Mode.....	53
ENC_Get_BSS_Parameter.....	54
ENC_Set_BSS_Parameter	55
ENC_Get_BSS_Parameter_AV	56
ENC_Set_BSS_Parameter_AV	57
ENC_Get_Overlay_Parameter.....	58
ENC_Overlay_Parameter	58
ENC_Set_Overlay_Parameter	59
ENC_Get_Video_Parameter	60
ENC_Set_Video_Parameter.....	62
ENC_Get_Video_Encode_Parameter	63
ENC_Set_Video_Encode_Parameter	65
ENC_Get_Video_Encode_Parameter_Ex.....	66
ENC_Set_Video_Encode_Parameter_Ex.....	68
ENC_Get_Audio_Format	69
ENC_Set_Audio_Format.....	70
ENC_Get_Audio_Parameter.....	71
ENC_Set_Audio_Parameter	72
ENC_Get_Audio_Encode_Parameter.....	73
ENC_Set_Audio_Encode_Parameter	75
ENC_Init_Movie	76
ENC_Record_Movie	77
ENC_Stop.....	78
ENC_Get_Record_Time.....	79
ENC_Set_Record_Time	80
ENC_Get_Movie_File	81
ENC_Set_Movie_File.....	82
ENC_Get_Frame_Count	83
ENC_Get_Time	84
ENC_Detect_Video_Input_Source.....	85
ENC_Set_Digital_Video_Input	86
ENC_Get_Digital_Video_Input.....	87

ENC_Set_Video_Encode_File.....	88
ENC_Get_Last_Error	89
VIDEO DECODER FUNCTIONS.....	90
DEC_Set_Callback	90
DEC_CB_STATUS	90
DEC_CB_ERROR	91
DEC_Can_Initialize.....	92
DEC_Get_Codec_Config	93
DEC_CONFIG.....	93
DEC_Set_Codec_Number	95
DEC_Initialize	96
DEC_Initialize_Ex.....	97
DEC_BSR_PARAMETER	97
DEC_CB_BSR.....	97
DEC_Terminate	99
DEC_Get_Media	100
DEC_Set_Media.....	101
DEC_Can_Playback.....	102
DEC_Get_Status.....	103
DEC_Can_Overlay_Window.....	104
DEC_Create_Overlay_Window	105
DEC_Destroy_Overlay_Window	106
DEC_Move_Overlay_Window	107
DEC_Resize_Overlay_Window	108
DEC_Show_Overlay_Window.....	109
DEC_Get_Overlay_Window.....	110
DEC_Get_Overlay_Rect.....	111
DEC_Set_Overlay_Rect.....	112
DEC_Start_Monitor	113
DEC_Stop_Monitor	114
DEC_Get_Monitor_Status	115
DEC_Get_BSR_Parameter	116
DEC_Set_BSR_Parameter.....	117
DEC_Get_Overlay_Parameter.....	118
DEC_Overlay_Parameter.....	118
DEC_Set_Overlay_Parameter	119
DEC_Get_Video_Parameter	120
DEC_Set_Video_Parameter.....	121
DEC_Get_Audio_Parameter.....	122
DEC_Set_Audio_Parameter	123
DEC_Get_Decompose_Parameter.....	124
DEC_Set_Decompose_Parameter	125
DEC_Play	126
DEC_Play_From	127
DEC_Pause	128
DEC_Resume.....	129
DEC_Stop.....	130
DEC_Get_Repeat.....	131
DEC_Set_Repeat	132
DEC_Get_Movie_File	133
DEC_Set_Movie_File.....	134
DEC_Get_Image_Size.....	135
DEC_Get_Frame_Count	136
DEC_Get_Time	137

DEC_Get_Type	138
DEC_Get_File_Type	139
DEC_Get_Playback_Time.....	140
DEC_Seek	141
DEC_Get_Sync_Stc_Value.....	142
DEC_Get_Last_Error	143
APPENDIX.....	144
Error Code summary	144

Preliminaries

New Functions

1. From Version 2.00

In the Version 2.00 update, the following functions have been changed with new arguments and added members. For details, please refer to the description for the functions.

Function	Changes
ENC_Set_Callback	Argument ENC_CB_TMAP has been changed to ENC_VOBU_ENT . Two arguments have been added.
ENC_Initialize	ENC_MEDIA has been deleted. ENC_Get_Media/ENC_Set_Media will replace ENC_Media
ENC_Can_Overlay_Window	One argument was added.
ENC_Create_Overlay_Window	Two arguments were added.
ENC_Get_Video_Encoder_Parameter_Ex	From the ENC_VIDEO_ENCODE_PARAMETER_EX structure, the reserved member was deleted, and 3 members were added.
ENC_Set_Video_Encoder_Parameter_Ex	Same as above.
ENC_Init_Movie	ENC_MODE was deleted.
DEC_Can_Overlay_Window	One argument was added.
DEC_Create_Overlay_Window	Two arguments were added.
DEC_Get_Decode_Parameter	Two members were added in DEC_DECODE_PARAMETER structure.
DEC_Set_Decode_Parameter	Same as above

2. From Version 2.10

In the Version 2.10 update, the following functions have been changed with new arguments and added members. For details, please refer to the description for the functions.

Function	Changes
DEC_Get_Decode_Parameter	One member was added to the DEC_DECODE_PARAMETER structure.
DEC_Set_Decode_Parameter	Same as above

* From this version, this SDK is created for Microsoft Visual C++ 6.0

3. From Version 3.00

In the Version 3.00 update, the following functions have been changed with new arguments and added members. For details, please refer to the description for the functions.

Function	Changes
ENC_Get_Video_Encoder_Parameter_Ex	One member was added to the ENC_Get_Video_Encoder_Parameter_Ex structure.
ENC_Set_Video_Encoder_Parameter_Ex	Same as above

* New APIs that have been added are not listed here.

4. From Version 3.10

In the Version 3.10 update, the following have been added.

[Video encoder function]

ENC_Get_Codec_Config	Get card information
ENC_Set_Codec_Number	Set the card to use by number
ENC_Get_Overlay_Rect	overlay window area acquisition
ENC_Set_Overlay_Rect	Set overlay window area
ENC_Set_Video_Encode_File	Set file for detailed video encode parameters

[Video decoder function]

DEC_Get_Codec_Config	Get card information
DEC_Set_Codec_Number	Set the card to use by number
DEC_Get_Overlay_Rect	overlay window area acquisition
DEC_Set_Overlay_Rect	Set overlay window area

5. From Version 3.22

In the Version 3.22 update, the following have been added.

[Video encoder function]

ENC_Get_BSS_Parameter_AV	Get the parameter for elementary stream memory transfer.
ENC_Set_BSS_Parameter_AV	Set the parameter for elementary stream memory transfer.
ENC_Set_Digital_Video_Input	Switch between analog video input and digital video input.
ENC_Get_Digital_Video_Input	Get the current video input status.

[Video decoder function]

DEC_Get_Sync_Stc_Value	Get the SYNC STC resistor value in HD814210
-------------------------------	---

Included Files

The **MVR-D2000 (Amber) Development Kit** provides a library for the development of custom applications to run on the MVR-D2000 (Amber) hardware.

The development kit contains the sample programs (both source code and executables) and includes the files listed below.

C/C++ include file

Program Files\Canopus\MVR-D2000 Development Kit\VC\Include\Mvrapidef.h

C/C++ library file

Program Files\Canopus\MVR-D2000 Development Kit\VC\Lib\Mvrapi.lib

C/C++ sample program files(source code)

Program Files\Canopus\MVR-D2000 Development Kit\VC\Samples\Encode
... encode
Program Files\Canopus\MVR-D2000 Development Kit\VC\Samples\Encode Memory
... encode (from memory)
Program Files\Canopus\MVR-D2000 Development Kit\VC\Samples\Com\
Encode Communication Ex
... encode (Communication)
Program Files\Canopus\MVR-D2000 Development Kit\VC\Samples\Decode
... decode
Program Files\Canopus\MVR-D2000 Development Kit\VC\Samples\Decode Memory
... decode (from memory)
Program Files\Canopus\MVR-D2000 Development Kit\VC\Samples\Com\
Decode Communication Ex
... decode (Communication)
Program Files\Canopus\MVR-D2000 Development Kit\VC\Samples\Multi
... Multiple boards support
Program Files\Canopus\MVR-D2000 Development Kit\VC\MvrCtrl
... ActiveX control

C/C++ sample program files(executable)

Program Files\Canopus\MVR-D2000\Encode.exe	...	encode
Program Files\Canopus\MVR-D2000\Encode_m.exe	...	encode (from memory)
Program Files\Canopus\MVR-D2000\Encode_cEx.exe	...	encode (Communication)
Program Files\Canopus\MVR-D2000\Decode.exe	...	decode
Program Files\Canopus\MVR-D2000\Decode_m.exe	...	decode (from memory)
Program Files\Canopus\MVR-D2000\Decode_cEx.exe	...	decode (Communication)
Program Files\Canopus\MVR-D2000\Multi.exe	...	Multiple board control
Program Files\Canopus\MVR-D2000\MvrCtrl.ocx	...	ActiveX control

Installation

For details on installing the development kit, please consult the Amber User's Manual. During installation, make sure to choose "Development Kit" from the component selection dialog box.

Visual C++ Application Development

Development Environment

Add the folders listed below to the path so the include and library files can be used in a project.

Note: It is assumed that the Development kit is installed under C:\Program Files\Canopus\MVR-D2000 Development Kit\

Include path

C:\Program Files\Canopus\MVR-D2000 Development Kit\VC\Include

Library path

C:\Program Files\Canopus\MVR-D2000 Development Kit\VC\Lib

Compile

When compiling include the file below.

Mvrapidef.h

Link

When linking include the file below.

Mvrapi.lib

Application Development in other environments

This Development Kit was created using Microsoft **Visual C++ 6.0**. It can be used in Microsoft Visual C++ 6.0 compatible environments.

We can not be sure of the results if other development environments are used.

If using another development environment, please consult the manufacturer for usage instructions with this development kit.

Sample Programs

There are some sample programs included in the development kit for reference. The sample programs were made in *Visual C++*. The included sample programs are listed in **C:\Program Files\Canopus\MVR-D2000 Development Kit\SAMPLES.TXT**.

Note: It is assumed that the Development kit is installed under C:\Program Files\Canopus\MVR-D2000 Development Kit\.

VC\Samples\Encode	Encode Program
VC\Samples\Encode Memory	Encode (from memory) Program
VC\Samples\COM\Encode Communication Ex	Encode (Communication) Program
VC\Samples\Decode	Decode Program
VC\Samples\Decode Memory	Decode (from memory) Program
VC\Samples\COM\Decode Communication Ex	Decode (Communication) Program

VC\Samples\Multi	Multiple board control
VC\Samples\MvrCtrl	ActiveX control

MFC is used to describe the *Visual C++* sample files.

Source Code

The source code included with the sample programs can be used or modified for incorporation into another application. However, there will be no support from Canopus for such software.

Canopus grants permission to use and modify the included source code. Distribution in any form of the included source code is prohibited. (Objects or complied applications can be distributed, but any source code material that contains source code included with this SDK may not be distributed.)

Canopus cannot be held responsible for any results of applications developed using the development kit or its included source code.

Questions

For questions related to the development kit, please contact Canopus technical support at the following:

E-mail amber_sdk@canopus.co.jp

When making an inquiry please be sure to include the following:

1. Module version information (in *Explorer* right click the file to view version information)
2. Development information
 - Compiler maker, version, language
 - Other software being used
3. Computer Environment
 - PC name and name of manufacturer
 - CPU name and type
 - Amount of memory installed
 - Windows NT information (Service Pack version, Service Pack 4 or higher is required)
 - Information on any peripherals (maker, model number)
 - Windows NT 3.5x is not recommended

Tutorial

Encoder Application Development

This chapter will explain how to create a video to MPEG2 format conversion application. All sample code is written in C.

The development of the application will follow this order.

- Selecting the card**
- Launching the encoder**
- Overlay window creation**
- Window position and size changing**
- Starting the encoder**
- Stopping the encoder**
- Overlay window interruption**
- Closing the encoder**

Note: File input/output connections are not explained in this section.

Selecting the card

When you have multiple MVR-D2000 (Amber) cards in the system, you need to check which cards are functional, and select which card you are going to use.

If you have only one card connected, or if you do not need to choose a card, this step is not required.

```
UINT enc_id;          // encoder identifier(global variable)
ENC_CONFIG config     // structure to get card information
int i ;

// Get the number of cards
memset(&config,0,sizeof(ENC_CONFIG));
ENC_Get_Codec_Config(enc_id,&config);

// look for a card that can be used
for(i=1;i<=config.NumberCodecs;i++)
{
    config.CodecNumber=i;
    ENC_Get_Codec_Config(enc_id,&config);
    if(config.CurrentUtilization==FALSE
    {
        if(config.CodecCaps&ENCCAP_ENCODE)
        {
            //Set the card to use
            ENC_Set_Codec_Number(enc_id,i);
            break;
        }
    }
}
```

```
}  
}  
}
```

Launching the encoder

To use the **MVR-D2000 (Amber)** encoding functions, it is necessary to notify the driver at application launch by calling the driver's **ENC_Initialize** or **ENC_Initialize_Ex** function.

Encoding to files

```
UINT enc_id;          // encoder identifier(global variable)  
ENC_MEDIA media;      // encoder type(global variable)  
  
ENC_RETURN      enc_return;  
  
enc_return = ENC_Initialize(enc_id);  
if (enc_return != ENC_SUCCESS)  
{  
    // cannot use the MVR-D2000 (Amber)  
  
    error handling  
}  
  
media = ENC_MEDIA_FILE;  
  
enc_return = ENC_Set_Media(enc_id, media);  
if (enc_return != ENC_SUCCESS)  
{  
    // cannot set the encode type  
  
    error handling  
}
```

Memory transfer

```
UINT enc_id;          // encoder identifier(global variable)  
ENC_MEDIA media;      // encoder type(global variable)  
  
ENC_RETURN      enc_return;  
ENC_BSS_PARAMETER bss_param;  
  
bss_param.fType = ENC_BSS_TYPE_PS;  
bss_param.pfnCallback = WriteProc; // Callback function  
bss_param.cbBuff = 0x8000; // buffer size  
bss_param.cBuff = 16; // buffer  
bss_param.dwData = (DWORD)INVALID_HANDLE_VALUE;
```

```
media=ENC_MEDIA_MEMORY;
enc_return = ENC_Initialize_Ex(enc_id, media, &bss_param);
if (enc_return != ENC_SUCCESS)
{
    // cannot use the MVR-D2000 (Amber)

    error handling
}

// callback function for memory transfer
void CALLBACK WriteProc(LPBYTE pbBuff, DWORD cbBuff, DWORD dwData)
{

    DWORD cbWritten;
    HANDLE hFile=(HANDLE)dwData;

    If(hFile!=INVALID_HANDLE_VALUE)

    {

        WriteFile(hFile, pbBuff, cbBuff, &cbWritten, NULL);

    }

}
```

Overlay window creation

After creating the application window the overlay display's capability will be checked, and if capable then the overlay window can use the client region.

```
UINT enc_id;          // encoder identifier(global variable)
UINT nWidth;          // overlay display width(global variable)
UINT nHeight;         // overlay display height(global variable)
HWND hWndApp;         // application main window handle(global variable)
HWND hWndOverlay;     // overlay window handle(global variable)
ENC_RETURN enc_return;

enc_return = ENC_Can_Overlay_Window(enc_id, nWidth, nHeight);
if (enc_return != ENC_SUCCESS)
{
    // overlay display cannot be used

    error handling
    return;
}

enc_return = ENC_Create_Overlay_Window(enc_id, hWndApp, &hWndOverlay, nWidth,
nHeight) ;
if (enc_return != ENC_SUCCESS)
{
```

```
        // overlay window creation failure

        error handling
        return;
    }
```

Changing position and size of the overlay window

When changing the position or size of the user's application window, the overlay window position and size must also change.

```
    UINT enc_id;          // encoder identifier(global variable)

    case WM_MOVE:
    {
        INT  xPos, yPos;

        xPos = (INT)(short) LOWORD(IParam);
        yPos = (INT)(short) HIWORD(IParam);
        ENC_Move_Overlay_Window(enc_id, xPos, yPos) ;
    }
        return 0;

    case WM_SIZE:
    {
        UINT  nWidth, nHeight;
        nWidth = (UINT)LOWORD(IParam);
        nHeight = (UINT)HIWORD(IParam);
        VERIFY(ENC_Resize_Overlay_Window(enc_id, nWidth, nHeight);
    }
        return 0;
```

Starting the encoder

To start the Encoder call the **ENC_Init_Movie** function. The Encoder will wait for the **ENC_Record_Movie** function to be called and then encoding will begin.

```
    UINT enc_id;          // encoder identifier(global variable)
    ENC_MEDIA media;       // encoder type(global variable)
    HANDLE hFile;          // file handle
    TCHAR szFile[MAX_PATH]; // file name
    ENC_BSS_PARAMETER bss_param;

    If(media==ENC_MEDIA_MEMORY)
    {
        hFile=CreateFile(szFile, GENERIC_READ|GENERIC_WRITE,
                        FILE_SHARE_READ,NULL,
                        CREATE_ALWAYS,
                        FILE_ATTRIBUTE_NORMAL|
```

```
        FILE_FLAG_SEQUENTIAL_SCAN,
        NULL);
ENC_Get_BSS_Parameter(enc_id,&bss_param);
Bss_param.dwData=(DWORD)hFile;
ENC_Set_BSS_Parameter(enc_id,&bss_param);

}

ENC_Init_Movie(enc_id);
ENC_Record_Movie(enc_id, ENC_RECORD_PS);
```

Stopping the Encoder

The Encoder can be stopped by calling the **ENC_Stop** function.

```
UINT enc_id;          // encoder identifier(global variable)

ENC_Stop(enc_id);
```

Overlay window interruption

Use **WM_DESTROY** to interrupt the overlay window.

```
UINT enc_id;          // encoder identifier(global variable)

ENC_Destroy_Overlay_Window(enc_id );
```

Closing the encoder

When closing the **MVR-D2000 (Amber)**'s application it is necessary to notify the driver by calling the **ENC_Terminate** function.

```
UINT enc_id;          // encoder identifier(global variable)

ENC_Terminate(enc_id );
```

Decoder Application Development

This chapter will explain how to create a MPEG2 format to video conversion application. All sample code is written in C.

The development of the application will follow this order.

- Selecting the card**
- Launching the decoder**
- Overlay window creation**
- Window position and size changing**
- Starting the decoder**
- Stopping the decoder**
- Overlay window interruption**
- Closing the decoder**

Note: File input/output connections are not explained in this section.

Selecting the card

When you have multiple MVR-D2000 (Amber) cards in the system, you need to check which cards are functional, and select which card you are going to use.

If you have only one card connected, or if you do not need to choose a card, this step is not required.

```
UINT dec_id;          // decoder identifier(global variable)
DEC_CONFIG config    // structure to get card information
int i;

// Get the number of cards
memset(&config,0,sizeof(DEC_CONFIG));
DEC_Get_Codec_Config(dec_id,&config);

// look for a card that can be used
for(i=1;i<=config.NumberCodecs;i++)
{
    config.CodecNumber=i;
    DEC_Get_Codec_Config(dec_id,&config);
    if(config.CurrentUtilization==FALSE
    {
        if(config.CodecCaps&DECCAP_DECODE)
        {
            //Set the card to use
            DEC_Set_Codec_Number(dec_id,i);
            break;
        }
    }
}
```


Launching the decoder

To use the **MVR-D2000 (Amber)** encoding functions, it is necessary to notify the driver at application launch by calling the driver's **DEC_Initialize** or **DEC_Initialize_Ex** function.

Decoding files

```
UINT dec_id;          // decoder identifier(global variable)
DEC_MEDIA media;      // decoder type(global variable)

DEC_RETURN      dec_return;

dec_return = DEC_Initialize(dec_id);
if (dec_return != DEC_SUCCESS)
{
    // cannot use the MVR-D2000 (Amber)

    error handling
}

media = DEC_MEDIA_FILE;

dec_return = DEC_Set_Media(dec_id, media);
if (dec_return != DEC_SUCCESS)
{
    // cannot set the decode type

    error handling
}
```

Memory transfer

```
UINT dec_id;          // decoder identifier(global variable)

DEC_RETURN      dec_return;
DEC_BSR_PARAMETER bsr_param;

bsr_param.fType = DEC_BSR_TYPE_PS;
bsr_param.pfnCallback = ReadProc; // Callback function
bsr_param.cbBuff = 0x8000; // buffer size
bsr_param.cBuff = 32; // buffer
bsr_param.dwData = (DWORD)INVALID_HANDLE_VALUE;

media=DEC_MEDIA_MEMORY;
dec_return = DEC_Initialize_Ex(dec_id, media, &bsr_param);
if (dec_return != DEC_SUCCESS)
{
    // cannot use the MVR-D2000 (Amber)

    error handling
}
```

```
// callback function for memory transfer
void CALLBACK ReadProc(LPBYTE pbBuff, DWORD cbWrite,
                      LPDWORD pcbWritten, DWORD dwData)
{
    HANDLE hFile=(HANDLE)dwData;

    If(hFile!=INVALID_HANDLE_VALUE)
    {
        ReadFile(hFile, pbBuff, cbWrite, pcbWritten, NULL);
    }
}
```

Overlay window creation

After creating the application window the overlay display's capability will be checked, and if capable then the overlay window can use the client region.

```
UINT dec_id;          // decoder identifier(global variable)
UINT nWidth;          // overlay display width(global variable)
UINT nHeight;         // overlay display height(global variable)
HWND hWndApp;         // application main window handle(global variable)
HWND hWndOverlay;     // overlay window handle(global variable)
DEC_RETURN dec_return;

dec_return = DEC_Can_Overlay_Window(dec_id, NULL, nWidth, nHeight);
if (dec_return != DEC_SUCCESS)
{
    // overlay cannot display

    error handling
    return;
}

dec_return = DEC_Create_Overlay_Window(dec_id, hWndApp, &hWndOverlay, 0, 0, nWidth,
nHeight) ;
if (dec_return != DEC_SUCCESS)
{
    // overlay window creation failure

    error handling
    return;
}
```

Changing position and size of the window

When changing the position or size of the user's application window, the overlay window position and size must also change.

```
UINT dec_id;          // decoder identifier(global variable)

case WM_MOVE:
{
    INT  xPos, yPos;

    xPos = (INT)(short) LOWORD(IParam);
    yPos = (INT)(short) HIWORD(IParam);
    DEC_Move_Overlay_Window(dec_id, xPos, yPos);
}
    return 0;

case WM_SIZE:
{
    UINT  nWidth, nHeight;

    nWidth = (UINT)LOWORD(IParam);
    nHeight = (UINT)HIWORD(IParam);
    VERIFY(DEC_Resize_Overlay_Window(dec_id, nWidth, nHeight);
}
    return 0;
```

Starting the decoder

To start decoding call the **DEC_Play** function.

```
UINT dec_id;          // decoder identifier(global variable)
DEC_MEDIA media; // decoder type
HANDLE hFile; // file handle
TCHAR szFile[MAX_PATH]; // file name

DEC_BSR_PARAMETER bsr_param;
if (media == DEC_MEDIA_MEMORY)
{
    hFile = CreateFile(szFile, GENERIC_READ,
        FILE_SHARE_READ, NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL |
        FILE_FLAG_NO_BUFFERING |
        FILE_FLAG_SEQUENTIAL_SCAN, NULL);
    DEC_Get_BSR_Parameter(dec_id, &bsr_param);
    bsr_param.dwData = (DWORD)hFile;
    DEC_Set_BSR_Parameter(dec_id, &bsr_param);
}

    DEC_Play(dec_id);
```

Stopping the decoder

The Decoder can be stopped by calling the **DEC_Stop** function.

```
UINT dec_id;          // decoder identifier(global variable)

DEC_Stop(dec_id);
```

Overlay window interruption

Using **WM_DESTROY** to interrupt the overlay window

```
UINT dec_id;          // decoder identifier(global variable)

DEC_Destroy_Overlay_Window(dec_id );
```

Closing the decoder

When closing the **MVR-D2000 (Amber)**'s application it is necessary to notify the driver by calling the **DEC_Terminate** function.

```
UINT dec_id;          // decoder identifier(global variable)

DEC_Terminate(dec_id );
```

Quick Reference

Video Encoder Functions

ENC_Set_Callback	callback function settings
ENC_Can_Initialize	check if the encoder can be initialized
ENC_Get_Codec_Config	Get card information
ENC_Set_Codec_Number	Set the card to use by number
ENC_Initialize	Encoder initialization
ENC_Initialize_Ex	Initialize extended encoder
ENC_Terminate	Encoder termination
ENC_Get_Media	Get encoder type
ENC_Set_Media	Set encoder type
ENC_Can_Record	Check if can record
ENC_Get_Status	Encoder status
ENC_Can_Overlay_Window	overlay window capability
ENC_Create_Overlay_Window	overlay window creation
ENC_Destroy_Overlay_Window	overlay window interruption
ENC_Move_Overlay_Window	overlay window move
ENC_Resize_Overlay_Window	overlay window resize
ENC_Show_Overlay_Window	overlay window display change
ENC_Get_Overlay_Window	overlay window handle acquisition
ENC_Get_Overlay_Rect	overlay window area acquisition
ENC_Set_Overlay_Rect	Set overlay window area
ENC_Start_Monitor	start monitor
ENC_Stop_Monitor	stop monitor
ENC_Get_Monitor_Status	inquire monitor status
ENC_Get_VideoCD_Mode	inquire Video CD mode
ENC_Set_VideoCD_Mode	specify Video CD mode
ENC_Get_BSS_Parameter	get parameter for memory transfer
ENC_Set_BSS_Parameter	set parameter for memory transfer
ENC_Get_BSS_Parameter_AV	Get the parameter for elementary stream memory transfer
ENC_Set_BSS_Parameter_AV	Set the parameter for elementary stream memory transfer
ENC_Get_Overlay_Parameter	get parameter for overlay
ENC_Set_Overlay_Parameter	set parameter for overlay
ENC_Get_Video_Parameter	inquire Video parameters
ENC_Set_Video_Parameter	specify Video parameters
ENC_Get_Video_Encode_Parameter	inquire Video encode parameters
ENC_Set_Video_Encode_Parameter	specify Video encode parameters
ENC_Get_Video_Encode_Parameter_Ex	inquire Extended Video encode parameters
ENC_Set_Video_Encode_Parameter_Ex	specify Extended Video encode parameters
ENC_Get_Audio_Format	inquire Audio format
ENC_Set_Audio_Format	specify Audio format
ENC_Get_Audio_Parameter	inquire Audio parameters
ENC_Set_Audio_Parameter	specify Audio parameters
ENC_Get_Audio_Encode_Parameter	inquire Audio encode parameters
ENC_Set_Audio_Encode_Parameter	specify Audio encode parameters
ENC_Init_Movie	Encoder start wait state
ENC_Record_Movie	Encoder start

ENC_Stop	Encoder stop
ENC_Get_Record_Time	Encode time acquisition
ENC_Set_Record_Time	Encode time settings
ENC_Get_Movie_File	Encode file acquisition
ENC_Set_Movie_File	Encode file settings
ENC_Get_Frame_Count	Encoded frame count acquisition
ENC_Get_Time	Encoded time acquisition
ENC_Detect_Video_Input_Source	Input source auto-detection
ENC_Set_Digital_Video_Input	Switch between analog video input and digital video input
ENC_Get_Digital_Video_Input	Get the current video input status
ENC_Set_Video_Encode_File	Set file for detailed video encode parameters
ENC_Get_Last_Error	get error

Video Decoder Functions

DEC_Set_Callback	callback function settings
DEC_Can_Initialize	check if the decoder can be initialized
DEC_Get_Codec_Config	Get card information
DEC_Set_Codec_Number	Set the card to use by number
DEC_Initialize	Decoder initialization
DEC_Initialize_Ex	Extended decoder initialization
DEC_Terminate	Decoder termination
DEC_Get_Media	get decode type
DEC_Set_Media	set decode type
DEC_Can_Playback	Check if can playback
DEC_Get_Status	inquire status
DEC_Can_Overlay_Window	overlay window capability check
DEC_Create_Overlay_Window	overlay window creation
DEC_Destroy_Overlay_Window	overlay window interruption
DEC_Move_Overlay_Window	overlay window move
DEC_Resize_Overlay_Window	overlay window resize
DEC_Show_Overlay_Window	overlay window display change
DEC_Get_Overlay_Window	inquire overlay window handle
DEC_Get_Overlay_Rect	overlay window area acquisition
DEC_Set_Overlay_Rect	Set overlay window area
DEC_Start_Monitor	start monitor
DEC_Stop_Monitor	stop monitor
DEC_Get_Monitor_Status	monitor status
DEC_Get_BSR_Parameter	get parameter for memory transfer
DEC_Set_BSR_Parameter	set parameter for memory transfer
DEC_Get_Overlay_Parameter	get parameter for overlay
DEC_Set_Overlay_Parameter	set parameter for overlay
DEC_Get_Video_Parameter	inquire video parameters
DEC_Set_Video_Parameter	specify video parameters
DEC_Get_Audio_Parameter	inquire audio parameter
DEC_Set_Audio_Parameter	specify audio parameters
DEC_Get_Decode_Parameter	inquire decode parameters
DEC_Set_Decode_Parameter	specify decode parameters
DEC_Play	Decoder start
DEC_Play_From	Start decode from set location
DEC_Pause	Decoder pause
DEC_Resume	Decoder resume
DEC_Stop	Decoder stop
DEC_Get_Repeat	inquire repeat state
DEC_Set_Repeat	specify repeat state
DEC_Get_Movie_File	inquire decode file
DEC_Set_Movie_File	specify decode file
DEC_Get_Image_Size	inquire image size
DEC_Get_Frame_Count	inquire decode frame count
DEC_Get_Time	inquire decode time
DEC_Get_Type	inquire decode type

DEC_Get_File_Type	inquire file type
DEC_Get_Playback_Time	get playback time
DEC_Seek	Set the position to start playback
DEC_Get_Sync_Stc_Value	Get the value from HD814210 SYNC STC register playback
DEC_Get_Last_Error	get error

Video Encoder Functions

ENC_Set_Callback

```
ENC_RETURN ENC_Set_Callback(UINT enc_id, ENC_CB_STATUS pStatus,  
                             ENC_CB_ERROR pError,  
                             ENC_CB_VOBU_ENT pVobu_ent,  
                             ENC_CB_S_PTM pSptm,  
                             ENC_CB_SVOB_ENT pSvob_ent);
```

Description:

Callback function settings.

Arguments:

enc_id	encoder identifier
pStatus	set the callback function to get internal status
pError	set the callback function for sending error
pVobu_ent	set the callback function for getting info in GOP mode
pSptm	reserved (set null)
pSvob_ent	reserved (set null)

Note:

Reference:

ENC_CB_STATUS

```
VOID CALLBACK StatusProc(UINT enc_id, ENC_STATUS_NOTIFY status);
```

Description:

Callback function to get the encoder's internal state.

Arguments:

enc_id	encoder identifier
status	internal state

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Status value	Explanation
ENC_NOTIFY_INITIALIZE	encoder was initialized
ENC_NOTIFY_TERMINATE	encoder was closed
ENC_NOTIFY_STOP	encoder was stopped
ENC_NOTIFY_INIT_MOVIE	encoder wait state started
ENC_NOTIFY_MOVIE	encode started

Reference:

ENC_CB_ERROR

VOID CALLBACK ErrorProc(UINT enc_id, ULONG error);

Description:

Callback function to get the operation error information.

Arguments:

enc_id	encoder identifier
error	error code(see Appendix)

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_CB_VOBU_ENT

VOID CALLBACK VobuProc(UINT enc_id, ENC_VONU_ENT_INFO*pVobu_ent, UINT Boundary_Flag);

Description:

Callback function to get the information in GOP mode.

Arguments:

enc_id	encoder identifier
pVobu_ent	pointer to ENC_VOBU_ENT_INFO structure
Boundary_Flag	returns 0 (always)

Note:

Reference:

ENC_VOBU_ENT_INFO

```
typedef struct {  
    USHORT Fstref_Sz;  
    USHORT Vobu_Pb_Tm;  
    ULONG Vobu_Sz;  
} ENC_VOBU_ENT_INFO;
```

Fstref_Sz	sector(PCK) number in the first I frame picture. Includes audio pack(A_PCK).
Vobu_Pb_Tm	total playback time(In fields)
Vobu_Sz	number of sectors(in PCK)

Description:

Stores info about the GOP mode.

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

The callback function set in **ENC_CB_ERROR** runs when an error occurs during encoding. If **ENC_FAIL** is returned when you run this function, please get the error info from **ENC_Get_Last_Error**.

Reference:

ENC_Get_Last_Error

ENC Can Initialize

ENC_RETURN ENC_CAN_Initialize(UINT enc_id);

Description:

Check if encoder can be Initialized.

Arguments:

enc_id	encoder identifier
--------	--------------------

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC Get Codec Config

```
ENC_RETURN ENC_Get_Codec_Config(UINT enc_id, ENC_CONFIG*pConfig);
```

Description:

Get card information

Arguments:

enc_id	encoder identifier
pConfig	pointer to ENC_CONFIG structure

Note:

If you set the **CodecNumber** in **ENC_CONFIG** to **0**, you can get the number of current cards in the system, and the card that is currently being used. Number of cards will be stored in **NumberCodecs**. The currently used card is stored in **CodecNumber**. If no cards are used, **0** will be stored in **CodecNumber**.

ENC CONFIG

```
typedef struct{
    UINT    NumberCodecs;
    UINT    CodecNumber;
    UINT    CodecCaps;
    UINT    CurrentUtilization;
}ENC_CONFIG;
```

Description:

get information about the card

Arguments:

NumberCodecs	stores the number of cards installed
CodecNumber	set 0 or the card number of the card you need the information of (1 – NumberCodecs)
CodecCaps	stores the information flag for the card set at CodecNumber

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Status value	Explanation
ENCCAP_NONE	no information
ENCCAP_ENCODE	flag to show encode is possible
ENCCAP_DECODE	flag to show decode is possible
ENCCAP_MONITOR	flag to show monitoring is possible

ENCCAP_AVI2MPEG

flag to show AVI to MPEG convert is possible

Arguments:

CurrentUtilization stores the info of the card stated at CodecNumber

Note:

Status value

TRUE

FALSE

Explanation

Used

Not used

ENC_Set_Codec_Number

```
ENC_RETURN ENC_Set_Codec_Number(UINT enc_id, UINT nNumber);
```

Description:

Set the card number of the card to be used

Arguments:

enc_id	encoder identifier
nNumber	card number

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Run ***ENC_Set_Codec_Number*** before initializing using ***ENC_Initialize*** or ***ENC_Initialize_Ex***. If you don't run ***ENC_Set_Codec_Number*** before initializing, a card will be selected automatically before initializing.

The card number is from **1** to the value of ***NumberCodecs***, in the ***ENC_CONFIG*** structure in the ***ENC_Get_Codec_Config***.

If the number is set to **0**, a card will be selected automatically.

Reference:

ENC_Get_Codec_Config, ***ENC_Initialize***, ***ENC_Initialize_Ex***

ENC Initialize

ENC_RETURN ENC_Initialize(UINT enc_id);

Description:

Initialize the encoder.

Arguments:

enc_id	encoder identifier
--------	--------------------

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC Initialize Ex

```
ENC_RETURN ENC_Initialize_Ex(UINT enc_id, ENC_MEDIA media,
                             ENC_BSS_PARAMETER* pParam);
```

Description:

Initialize the encoder, and set the media and memory transfer settings.

Arguments:

enc_id	encoder identifier
media	encoder type

Status value

ENC_MEDIA_FILE
ENC_MEDIA_MEMORY

Explanation

encode to file
transfer to memory

pParam	The pointer pParam for ENC_BSS_PARAMETER structure, that sets the parameter for memory transfer, is set when media is ENC_MEDIA_MEMORY . Otherwise, NULL is returned.
--------	--

ENC BSS PARAMETER

```
typedef struct {
    ENC_BSS_TYPE fType;
    ENC_CB_BSS pfnCallback;
    DWORD cbBuff;
    DWORD cBuff;
    DWORD dwData;
} ENC_BSS_PARAMETER;
```

fType	reserved (returns ENC_BSS_TYPE_PS)
pfnCallback	set callback function for memory transfer
cbBuff	buffer size for memory transfer during callback. Size is increment of pack size (2048 bytes)
cBuff	size of buffer set in cbBuff
dwData	32 bit data sent during callback

Description:

Set the parameter info for memory transfer.

ENC CB BSS

```
VOID CALLBACK BssProc(LPBYTE pbBuff, DWORD cbBuff, DWORD dwData);
```

pBuff	pointer to buffer where the encode data is stored
cbBuff	encode data size
dwData	value of dwData set in ENC_BSS_PARAMETER structure

Description:

Callback function for transferring encoded data.

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Memory transfer can not be done in **Video CD** mode

Reference:

ENC_Set_VideoCD_Mode

ENC Terminate

ENC_RETURN ENC_Terminate(UINT enc_id);

Description:

Close encoder.

Arguments:

enc_id	encoder identifier
--------	--------------------

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Executing **ENC_Terminate** cancels all settings using **ENC_Set_Callback**.

Reference:

ENC_Set_Callback

ENC_Get_Media

ENC_RETURN ENC_Get_Media(UINT enc_id, ENC_MEDIA* media);

Description:

Get the encode type.

Arguments:

enc_id	encoder identifier
media	pointer to ENC_MEDIA (Get encode type)

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Reference:

ENC_Initialize_Ex

ENC_Set_Media

ENC_RETURN ENC_Set_Media(UINT enc_id, ENC_MEDIA media);

Description:

Set the encode type.

Arguments:

enc_id	encoder identifier
media	encode type

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Reference:

ENC_Initialize_Ex

ENC Can Record

ENC_RETURN ENC_Can_Record(UINT enc_id);

Description:

Check if encoding is possible.

Arguments:

enc_id	encoder identifier
--------	--------------------

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Reference:

ENC_Get_Status

ENC_RETURN ENC_Get_Status(UINT enc_id, ENC_STATUS* status);

Description:

Get the encoder's internal state.

Arguments:

enc_id	encoder identifier
status	pointer to ENC_STATUS

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Status value	Explanation
ENC_STATUS_WAITINITIALIZE	initialize wait state
ENC_STATUS_WAITTERMINATE	terminate wait state
ENC_STATUS_STOP	stopped
ENC_STATUS_MOVIE	encoding

Reference:

ENC Can Overlay Window

ENC_RETURN ENC_Can_Overlay_Window(UINT enc_id, HANDLE hMonitor, UINT nWidth, UINT nHeight);

Description:

Specifies if the overlay window be used.

Arguments:

enc_id	encoder identifier
hMonitor	monitor handle
nWidth	overlay window width
nHeight	overlay window height

Return values:

ENC_SUCCESS	if the overlay window can be used
ENC_FAIL	if the overlay window can not be used

Note:

In **hMonitor**, set the monitor handle (primary or secondary) in multi monitor systems. If not multi monitor, set **NULL** in **hMonitor**.

Reference:

ENC Create Overlay Window

ENC_RETURN ENC_Create_Overlay_Window(UINT enc_id, HWND hWndParent, HWND* hWndOverlay,
int x, int y, UINT nWidth, UINT nHeight);

Description:

Create an overlay window.

Arguments:

enc_id	encoder identifier
hWndParent	parent window handle
hWndOverlay	overlay window handle address
x	overlay window position (left)
y	overlay window position (top)
nWidth	overlay window width (Max. 720)
nHeight	overlay window height (Max. NTSC 480, PAL 576)

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

When creating the overlay window the maximum size limitations apply. After creating the overlay window, it can be resized above those limitations by using **ENC_Resize_Overlay_Window**. The Overlay window can not be created during encoding.

Reference:

ENC_Resize_Overlay_Window

ENC_Destroy_Overlay_Window

ENC_RETURN ENC_Destroy_Overlay_Window(UINT enc_id);

Description:

Destroy the overlay window.

Arguments:

enc_id	encoder identifier
--------	--------------------

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Cannot interrupt overlay window while encoding.

Reference:

ENC Move Overlay Window

ENC_RETURN ENC_Move_Overlay_Window(UINT enc_id, INT x, INT y);

Description:

Move the overlay window.

Arguments:

enc_id	encoder identifier
x	client region of parent window upper left corner X coordinate
y	client region of parent window upper right corner Y coordinate

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC Resize Overlay Window

ENC_RETURN ENC_Resize_Overlay_Window(UINT enc_id, UINT nWidth, UINT nHeight);

Description:

Resize the overlay window.

Arguments:

enc_id	encoder identifier
nWidth	overlay window width
nHeight	overlay window height

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC Show Overlay Window

ENC_RETURN ENC_Show_Overlay_Window(UINT enc_id, BOOL fShow);

Description:

Change the overlay window display state.

Arguments:

enc_id	encoder identifier
fShow	display the overlay window (TRUE), do not display the overlay window (FALSE)

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Get_Overlay_Window

ENC_RETURN ENC_Get_Overlay_Window(UINT enc_id, HWND* pWnd);

Description:

Get the overlay window handle.

Arguments:

enc_id	encoder identifier
pWnd	pointer to overlay window handle

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Get_Overlay_Rect

ENC_RETURN ENC_Get_Overlay_Rect(UINT enc_id, LPRECT pRect);

Description:

Get the overlay window area info.

Arguments:

enc_id	encoder identifier
pRect	pointer to RECT structure to get the overlay window area info

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Set_Overlay_Rect

ENC_RETURN ENC_Set_Overlay_Rect(UINT enc_id, LPRECT pRect);

Description:

Set the overlay window area.

Arguments:

enc_id	encoder identifier
pRect	pointer to RECT structure to get the overlay window area info

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

The area that can be set with **pRect** is as below;

Video Format	Horizontal Position	Vertical Position
NTSC	0 to 720	0 to 480
PAL	0 to 720	0 to 576

This function can not be set while encoding.

ENC_Start_Monitor

ENC_RETURN ENC_Start_Monitor(UINT enc_id);

Description:

Start the monitor.

Arguments:

enc_id	encoder identifier
--------	--------------------

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

When using **ENC_Start_Monitor** the overlay window returns to the size at time of creation.
Cannot be started during encoding.

Reference:

ENC_Stop_Monitor

ENC_RETURN ENC_Stop_Monitor(UINT enc_id);

Description:

Stop the monitor.

Arguments:

enc_id	encoder identifier
--------	--------------------

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Cannot be stopped during encoding.

Reference:

ENC_Get_Monitor_Status

ENC_RETURN ENC_Get_Monitor_Status(UINT enc_id, UINT* monitor);

Description:

Get the monitor status.

Arguments:

enc_id	encoder identifier
monitor	monitor status – TRUE = Monitoring, FALSE = Not monitoring

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Get_VideoCD_Mode

ENC_RETURN ENC_Get_VideoCD_Mode(UINT enc_id, LPINT lpbEnable);

Description:

Get the Video CD mode setting.

Arguments:

enc_id	encoder identifier
lpbEnable	pointer to Video CD mode – TRUE = Video CD, FALSE = Non-Video CD

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Set_VideoCD_Mode

ENC_Set_VideoCD_Mode

ENC_RETURN ENC_Set_VideoCD_Mode(UINT enc_id, INT bEnable);

Description:

Set the Video CD mode.

Arguments:

enc_id	encoder identifier
bEnable	enable the Video CD mode

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note: When selecting Video CD mode the following settings are used.

Item	Value
MPEG type	MPEG1
Video input format	SIF
Video input size	NTSC = 352×240, PAL = 352×288
Video bit rate	1150000 bps
CBR/VBR	CBR
Audio bit rate	224000 bps
Sampling rate	44100 Hz
Layer	Layer2
Audio mode module	Other than Mono (will be set to stereo)
Emphasis	CCITT (CCITT can also be turned off.)
GOP mode	set to GOP non-completed mode

Reference:

ENC_Get_VideoCD_Mode, ENC_Get_Video_Encode_Parameter, ENC_Get_Audio_Parameter,
ENC_Get_Audio_Encode_Parameter

ENC Get BSS Parameter

ENC_RETURN ENC_Get_BSS_Parameter(UINT enc_id, ENC_BSS_PARAMETER*pParam);

Description:

Get the parameter for memory transfer.

Arguments:

enc_id	encoder identifier
pParam	pointer to ENC_BSS_PARAMETER structure (to get memory transfer parameter)

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Initialize_Ex

ENC_Set_BSS_Parameter

ENC_RETURN ENC_Set_BSS_Parameter(UINT enc_id, ENC_BSS_PARAMETER*pParam);

Description:

Set the parameter for memory transfer.

Arguments:

enc_id	encoder identifier
pParam	pointer to ENC_BSS_PARAMETER structure (to set memory transfer parameter)

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Initialize_Ex

ENC_Get_BSS_Parameter_AV

ENC_RETURN **ENC_Get_BSS_Parameter_AV**(UINT enc_id, ENC_BSS_PARAMETER*pParamAudio , ENC_BSS_PARAMETER*pParamVideo);

Description:

Get the parameter for elementary stream memory transfer.

Arguments:

enc_id	encoder identifier
pParamAudio	pointer to ENC_BSS_PARAMETER structure (to get the audio memory transfer parameter)
pParamVideo	pointer to ENC_BSS_PARAMETER structure (to get the video memory transfer parameter)

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

The fType member in ENC_BSS_PARAMETER is not used during elementary stream transfer.

Reference:

ENC_Initialize_Ex

ENC_Set_BSS_Parameter_AV

ENC_RETURN **ENC_Set_BSS_Parameter_AV**(UINT enc_id, ENC_BSS_PARAMETER*pParamAudio , ENC_BSS_PARAMETER*pParamVideo);

Description:

Get the parameter for elementary stream memory transfer.

Arguments:

enc_id	encoder identifier
pParamAudio	pointer to ENC_BSS_PARAMETER structure (to get the audio memory transfer parameter)
pParamVideo	pointer to ENC_BSS_PARAMETER structure (to get the video memory transfer parameter)

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

The fType member in ENC_BSS_PARAMETER is not used during elementary stream transfer.

Reference:

ENC_Initialize_Ex

ENC Get Overlay Parameter

ENC_RETURN ENC_Get_Overlay_Parameter(UINT enc_id, ENC_OVERLAY_PARAMETER*pParam);

Description:

Get the parameter for displaying overlay.

Arguments:

enc_id	encoder identifier
pParam	pointer to ENC_OVERLAY_PARAMETER structure (to get overlay parameter)

ENC Overlay Parameter

```
typedef struct {  
    UINT nOverlayBrightness;  
    int nOverlayContrast;  
    int nOverlaySaturation;  
} ENC_OVERLAY_PARAMETER;
```

Description:

Stores the parameter for displaying overlay.

nOverlayBrightness	brightness value ENCMIN_VIDEO_OVERLAY_BRIGHTNESS = minimum ENCMAX_VIDEO_OVERLAY_BRIGHTNESS = Maximum ENCDEF_VIDEO_OVERLAY_BRIGHTNESS = default
nOverlayContrast	contrast value ENCMIN_VIDEO_OVERLAY_CONTRAST = minimum ENCMAX_VIDEO_OVERLAY_CONTRAST = Maximum ENCDEF_VIDEO_OVERLAY_CONTRAST = default
nOverlaySaturation	saturation value ENCMIN_VIDEO_OVERLAY_SATURATION = minimum ENCMAX_VIDEO_OVERLAY_SATURATION = Maximum ENCDEF_VIDEO_OVERLAY_SATURATION = default

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

ENC_Set_Overlay_Parameter

ENC_RETURN ENC_Set_Overlay_Parameter(UINT enc_id, ENC_OVERLAY_PARAMETER*pParam);

Description:

Set the parameter for overlay.

Arguments:

enc_id	encoder identifier
pParam	pointer to ENC_OVERLAY_PARAMETER structure (to set overlay parameter)

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Get_Overlay_Parameter

ENC Get Video Parameter

ENC_RETURN ENC_Get_Video_Parameter(UINT enc_id, ENC_VIDEO_PARAMETER* Video_Param);

Description:

Get the video parameters.

Arguments:

enc_id	encoder identifier
Video_Param	pointer to a ENC_VIDEO_PARAMETER structure

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

ENC_VIDEO_PARAMETER structure definition

```
typedef struct {
    int    TV_System;
    int    fInputSource;
    UINT   nInputBrightness;
    int    nInputContrast;
    int    nInputHue;
    int    nInputSaturation;
    BOOL   fOutputMonitor;
} ENC_VIDEO_PARAMETER;
```

Value

TV_System

ENC_VIDEO_TV_SYSTEM_NTSC
ENC_VIDEO_TV_SYSTEM_PAL

Explanation

Broadcast standard settings.

NTSC
PAL

fInputSource

ENC_VIDEO_INPUT_SOURCE_COMPOSITE
ENC_VIDEO_INPUT_SOURCE_SVIDEO

Input source settings.

Composite
S-Video

nInputBrightness

ENCMIN_VIDEO_INPUT_BRIGHTNESS
ENCMAx_VIDEO_INPUT_BRIGHTNESS
ENCDEF_VIDEO_INPUT_BRIGHTNESS

Video input brightness settings.

minimum value
maximum value
default value

nInputContrast

ENCMIN_VIDEO_INPUT_CONTRAST
ENCMAx_VIDEO_INPUT_CONTRAST
ENCDEF_VIDEO_INPUT_CONTRAST

Video input contrast.

minimum value
maximum value
default value

nInputHue

ENCMIN_VIDEO_INPUT_HUE
ENCMAx_VIDEO_INPUT_HUE

Video input hue.

minimum value
maximum value

ENCDEF_VIDEO_INPUT_HUE	default value
nInputSaturation	Video input saturation.
ENCMIN_VIDEO_INPUT_SATURATION	minimum value
ENCMAX_VIDEO_INPUT_SATURATION	maximum value
ENCDEF_VIDEO_INPUT_SATURATION	default value
fOutputMonitor	Video output.
TRUE	video outputON
FALSE	video outputOFF

Reference:

ENC_Set_Video_Parameter

ENC_RETURN ENC_Set_Video_Parameter(UINT enc_id, ENC_VIDEO_PARAMETER* Video_Param);

Description:

Set the Video parameter settings.

Arguments:

enc_id	encoder identifier
Video_Param	pointer to a ENC_VIDEO_PARAMETER structure

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

If changing the **TV_System**, **fInputSource**, **fOutputMonitor** values of **ENC_VIDEO_PARAMETER** during monitoring then stop the monitor by using **ENC_Stop_Monitor** before changing the values. To resume use **ENC_Start_Monitor**.

Reference:

ENC_Stop_Monitor, **ENC_Start_Monitor**, **ENC_Get_Video_Parameter**

ENC_Get_Video_Encode_Parameter

```
ENC_RETURN ENC_Get_Video_Encode_Parameter(UINT enc_id,
                                           ENC_VIDEO_ENCODE_PARAMETER* Video_Param);
```

Description:

Get the video encode parameters.

Arguments:

enc_id	encoder identifier
Video_Param	pointer to a ENC_VIDEO_ENCODE_PARAMETER structure

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

ENC_VIDEO_ENCODE_PARAMETER structure definition

```
typedef struct {
    int      V_CodingMode;
    DWORD fProfileAndLevel;
    DWORD fConvertType;
    WORD   wImageLeft;
    WORD   wImageTop;
    WORD   wImageWidth;
    WORD   wImageHeight;
    DWORD dwBitrate;
    int    EncodeType;
    int    GopPattern;
} ENC_VIDEO_PARAMETER;
```

Value**Explanation****V_CodingMode**

ENCVEPARAM_V_CODING_MODE_MPEG1
ENCVEPARAM_V_CODING_MODE_MPEG2

MPEG type

MPEG1
MPEG2

fProfileAndLevel

ENCVEPARAM_PROFILE_AND_LEVEL_MP_ML
ENCVEPARAM_PROFILE_AND_LEVEL_MP_LL
ENCVEPARAM_PROFILE_AND_LEVEL_SP_ML

Profile and Level settings

MP@ML
MP@LL
SP@ML

fConvertType

ENCVEPARAM_CONVERT_TYPE_STANDARD
ENCVEPARAM_CONVERT_TYPE_SIF
ENCVEPARAM_CONVERT_TYPE_HALF_D1

video format settings

Standard width1 × height1
SIF width1/2 × height1/2
Half-D1 width1/2 × height1

wImageLeft**wImageTop****wImageWidth****reserved 0****reserved 0**

video input width in increments of 16 pixels.

wImageHeight			video input height in increments of 16 pixels.	
MPEG TYPE	LEVEL	Format	Size(NTSC)	SIZE(PAL)
MPEG 1	-	SIF	352x240	352x288
MPEG 2	MP@ML	Standard	720x480	720x576
MPEG 2	MP@ML	SIF	352x240	352x288
MPEG 2	MP@ML	Half-D1	352x480	352x576
MPEG 2	MP@LL	SIF	352x240	352x288
MPEG 2	SP@ML	Standard	720x480	720x576
MPEG 2	SP@ML	SIF	352x240	352x288
MPEG 2	SP@ML	Half-D1	352x480	352x576
dwBitrate			bit rate(bps) in 400bps denominations.	
MPEG TYPE			BITRATE	
MPEG 1			1,000,000 – 1,800,000 bps	
MPEG 1 (Video CD)			1,150,000 only	
MPEG 2 (MP@ML)			2,000,000 – 15,000,000 bps	
MPEG 2 (MP@LL)			2,000,000 – 4,000,000 bps	
MPEG 2 (SP@ML)			2,000,000 – 15,000,000 bps	
EncodeType			CBR/VBR selection	
ENCVEPARAM_ENCODE_TYPE_CBR			CBR	
ENCVEPARAM_ENCODE_TYPE_CBR			VBR	
GopPattern			GOP pattern selection	
ENCVEPARAM_GOP_PATTERN_IFRAME			I Frame	
ENCVEPARAM_GOP_PATTERN_IBBP			IBBP	

Reference:

ENC_Set_Video_Encode_Parameter

```
ENC_RETURN ENC_Set_Video_Encode_Parameter(UINT enc_id,  
                                           ENC_VIDEO_ENCODE_PARAMETER* Video_Param);
```

Description:

Set the video encoding parameters.

Arguments:

enc_id	encoder identifier
Video_Param	pointer to a ENC_VIDEO_PARAMETER structure

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

By using the **ENC_Set_Video_Encode_Parameter** the following values can be set automatically.

Item	Value		
	I Frame	IBBP Other than SP@ML	SP@ML
GOP number of pictures	1	15	15
GOP I frame and P frame frequency	1	3	1
VBR peak bit rate	video bit rate 2x	current setting	current setting
Closed GOP	OFF	OFF	OFF
Aspect ratio	4:3	4:3	4:3

Reference:

ENC_Get_Video_Encode_Parameter, ENC_Get_Video_Encode_Parameter_Ex

ENC_Get_Video_Encode_Parameter_Ex

```
ENC_RETURN ENC_Get_Video_Encode_Parameter_Ex(UINT enc_id,
                                             ENC_VIDEO_ENCODE_PARAMETER_EX* Video_Param);
```

Description:

Get the extended video parameters.

Arguments:

enc_id	encoder identifier
Video_Param	pointer to a ENC_VIDEO_PARAMETER_EX structure

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

These parameters will be extended as necessary.

ENC_VIDEO_ENCODE_PARAMETER_EX structure definitions

```
typedef struct {
    ENC_VIDEO_ENCODE_PARAMETER EncodeParam;
    WORD cPictureInterval;
    WORD cPicturePerGop;
    WORD dwVbrPeak;
    BOOL fClosedGop;
    BOOL fGopComplete;
    WORD wAspectRatio;
    BOOL fLowDelay
} ENC_VIDEO_ENCODE_PARAMETER_EX;
```

Value

EncodeParam

Explanation

Extended video encode parameters

cPictureInterval

ENCVEPARAMEX_MIN_PICTURE_INTERVAL
ENCVEPARAMEX_MAX_PICTURE_INTERVAL

GOP I frame, P frame frequency

minimum value
maximum value

cPicturePerGop

ENCVEPARAMEX_MIN_PICTURE_PER_GOP
ENCVEPARAMEX_MAX_PICTURE_PER_GOP

GOP number of pictures per group

minimum value
maximum value

dwVbrPeak

MPEG TYPE
MPEG 1
MPEG 1 (Video CD)
MPEG 2 (MP@ML)
MPEG 2 (MP@LL)
MPEG 2 (SP@ML)

VBR peak bit rate

PEAK BITRATE
Bit rate setting - 1,800,000 bps
Can't set
Bit rate setting - 15,000,000 bps
Bit rate setting - 4,000,000 bps
Bit rate setting - 15,000,000 bps

fClosedGop

TRUE
FALSE

Closed GOP setting

ON
OFF

fGopComplete

TRUE
FALSE

GOP complete mode setting

GOP complete mode
GOP non-complete mode

wAspectRatio

ENCVEPARAMEX_ASPECT_RATIO_4_3
ENCVEPARAMEX_ASPECT_RATIO_16_9

Aspect ratio

4:3
16:9

fLowDelay

TRUE
FALSE

Low delay setting

ON
OFF

Reference:

ENC_Get_Video_Encode_Parameter

ENC_Set_Video_Encode_Parameter_Ex

```
ENC_RETURN ENC_Set_Video_Encode_Parameter_Ex(UINT enc_id,
                                             ENC_VIDEO_ENCODE_PARAMETER_EX* Video_Param);
```

Description:

Set the extended Video Encode parameters.

Arguments:

enc_id	encoder identifier
Video_Param	pointer to a ENC_VIDEO_ENCODE_PARAMETER_EX structure

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

If you set the **fGopComplete** in **ENC_VIDEO_ENCODE_PARAMETER_EX** to GOP complete mode, the below values will be set.

Layer	LAYER II
Sampling rate	48kHz

If you set the **wAspectRatio** in **ENC_VIDEO_ENCODE_PARAMETER_EX** to **TRUE**, the display area size information will be added to the video elementary stream. This information will not be added in **MPEG1**.

Video Format	Size	display area size
NTSC Standard	720x480	540x480
NTSC SIF	352x240	264x240
NTSC Half-D1	352x480	264x480
PAL Standard	720x576	540x576
PAL SIF	352x288	264x288
PAL Half-D1	352x576	264x576

If you set the **fLowDelay** in **ENC_VIDEO_ENCODE_PARAMETER_EX** to **ON**, the **cPictureInterval** in the structure will be set in **ENCVEPARAMEX_MIN_PICTURE_INTERVAL**.

Reference:

ENC_Get_Video_Encode_Parameter, **ENC_Get_Video_Encode_Parameter_Ex**,
ENC_Get_Audio_Parameter, **ENC_Get_Audio_Encode_Parameter**

ENC_Get_Audio_Format

ENC_RETURN ENC_Get_Audio_Format (UINT enc_id, ENC_AUDIO_FORMAT *pAudioFormat);

Description:

Get the audio format.

Arguments:

enc_id	encoder identifier
pAudioFormat	pointer to the audio format address

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

AudioFormat Value	Explanation
ENC_AUDIO_FORMAT_MPEG	MPEG
ENC_AUDIO_FORMAT_PCM_MONO	PCM monaural
ENC_AUDIO_FORMAT_PCM_STEREO	PCM stereo

Reference:

ENC_Set_Audio_Format

ENC_RETURN ENC_Set_Audio_Format (UINT enc_id, ENC_AUDIO_FORMAT AudioFormat);

Description:

Set the audio format.

Arguments:

enc_id	encoder identifier
AudioFormat	audio format setting

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Get_Audio_Format

ENC_Get_Audio_Parameter

```
ENC_RETURN ENC_Get_Audio_Parameter(UINT enc_id, ENC_AUDIO_PARAMETER*
                                   Audio_Param);
```

Description:

Get the audio parameters.

Arguments:

enc_id	encoder identifier
Audio_Param	pointer to ENC_AUDIO_PARAMETER structure

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

ENC_AUDIO_PARAMETER structure definition

```
typedef struct {
    DWORD          dwSamplingFrequency;
    BOOL           fOutputMonitor;
    BOOL           fOutputMute;
    DWORD          dwLeftOutputAtt;
    DWORD          dwRightOutputAtt;
    int            nLeftInputGain;
    int            nRightInputGain;
} ENC_AUDIO_PARAMETER;
```

Value	Explanation
dwSamplingFrequency	Sampling rate (Hz)
fOutputMonitor	audio output state
TRUE	audio output ON
FALSE	audio output OFF
fOutputMute	mute state
TRUE	mute ON
FALSE	mute OFF
dwLeftOutputAtt	left volume
ENCMIN_AUDIO_OUTPUT_ATT	minimum value
ENCMAX_AUDIO_OUTPUT_ATT	maximum value
dwRightOutputAtt	right volume (same value as left)
nLeftInputGain	gain to left audio PCM input
ENCMIN_AUDIO_INPUT_GAIN	minimum value
ENCMAX_AUDIO_INPUT_GAIN	maximum value
nRightInputGain	gain to right audio PCM input (value same as left)

ENC_Set_Audio_Parameter

```
ENC_RETURN ENC_Set_Audio_Parameter(UINT enc_id, ENC_AUDIO_PARAMETER*
                                   Audio_Param);
```

Description:

Set the audio parameters.

Arguments:

enc_id	encoder identifier
Audio_Param	pointer to ENC_AUDIO_PARAMETER structure

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

To change the `ENC_AUDIO_PARAMETER` `dwSamplingFrequency` or `fOutputMonitor` during monitoring, stop the monitor using `ENC_Stop_Monitor`. Use `ENC_Set_Audio_Parameter`. `ENC_Start_Monitor` to start the monitor again.

Reference:

`ENC_Stop_Monitor`, `ENC_Start_monitor`, `ENC_Get_Audio_Parameter`

ENC Get Audio Encode Parameter

```
ENC_RETURN ENC_Get_Audio_Encode_Parameter(UINT enc_id,
                                           ENC_AUDIO_ENCODE_PARAMETER* Audio_Param);
```

Description:

Get the audio encode parameters.

Arguments:

enc_id	encoder identifier
Audio_Param	pointer to ENC_AUDIO_PARAMETER structure

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

ENC_AUDIO_ENCODE_PARAMETER structure definition

```
typedef struct {
    DWORD          nLayer;
    BOOL           fProtection;
    int            AudioBitrate;
    int            Channel_Num;
    DWORD          fModeEx;
    BOOL           fCopyright;
    BOOL           fOriginal;
    BOOL           fEmphasis;
} ENC_AUDIO_ENCODE_PARAMETER;
```

Value	Explanation
nLayer	reserved, ENCAEPARAM_LAYER2
fProtection	protection status
TRUE	error check on
FALSE	error check off
AudioBitrate	Audio bit rate
Channel_Num	Channel status
ENCAEPARAM_CHANNEL_NUM_STEREO	stereo
ENCAEPARAM_CHANNEL_NUM_MONO	monaural
ENCAEPARAM_CHANNEL_NUM_DUAL_MONO	dual channel
ENCAEPARAM_CHANNEL_NUM_JOINT_STEREO	joint stereo
fModeEx	reserved
fCopyright	Copyright status
TRUE	copyright
FALSE	no copyright

fOriginal	original/copy status
TRUE	original
FALSE	copy
fEmphasis	emphasis mode status
ENCAEPARAM_EMPHASIS_NONE	none
ENCAEPARAM_EMPHASIS_5015	50/15 us
ENCAEPARAM_EMPHASIS_CCITT	CCITT J.17

Reference:

ENC_Set_Audio_Encode_Parameter

```
ENC_RETURN ENC_Set_Audio_Encode_Parameter(UINT enc_id,  
                                           ENC_AUDIO_ENCODE_PARAMETER* Audio_Param);
```

Description:

Set the audio encoder parameters.

Arguments:

enc_id	encoder identifier
Audio_Param	pointer to ENC_AUDIO_ENCODE_PARAMETER structure

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Get_Audio_Encode_Parameter

ENC_Init_Movie

ENC_RETURN ENC_Init_Movie(UINT enc_id);

Description:

Encoder is put into wait state.

Arguments:

enc_id	encoder identifier
--------	--------------------

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Record_Movie

ENC_RETURN ENC_Record_Movie(UINT enc_id, UINT enc_mode_flag);

Description:

Start encoding.

Arguments:

enc_id	encoder identifier
enc_mode_flag	encode mode setting

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

value	explanation
enc_mode_flag	encode mode setting
ENC_RECORD_AUDIO	audio only
ENC_RECORD_VIDEO	video only (elementary stream)
ENC_RECORD_AUDIO ENC_RECORD_VIDEO	audio + video
ENC_RECORD_PS	MPEG1-system stream, MPEG2-program stream

If you run ENC_Initialize in memory transfer mode, the following encoding cannot be done:

1. Audio + Video encode
2. When audio is PCM, audio only encode

Reference:

ENC_Initialize, ENC_Initialize_Ex, ENC_Get_Audio_Format, ENC_Set_Audio_Format

ENC_Stop

ENC_RETURN ENC_Stop(UINT enc_id);

Description:

Stop encoding.

Arguments:

enc_id	encoder identifier
--------	--------------------

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC Get Record Time

ENC_RETURN ENC_Get_Record_Time(UINT enc_id, UINT* rec_time, UINT* enabled);

Description:

Get the time to be encoded.

Arguments:

enc_id	encoder identifier
rec_time	pointer to encode time
enabled	pointer to encode max time status

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Set_Record_Time

ENC_Set_Record_Time

ENC_RETURN ENC_Set_Record_Time(UINT enc_id, UINT* rec_time, UINT* enabled);

Description:

Set the time to be encoded.

Arguments:

enc_id	encoder identifier
rec_time	encode time (seconds)
enabled	encode time limit flag

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Value	Explanation
TRUE	time limit
FALSE	no time limit

Reference:

ENC_Get_Movie_File

ENC_RETURN ENC_Get_Movie_File(UINT enc_id, UINT enc_mode_flag, LPTSTR enc_file);

Description:

Get the movie filename.

Arguments:

enc_id	encoder identifier
enc_mode_flag	encode file type
enc_file	address of the encode file name

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Set_Movie_File

ENC_Set_Movie_File

```
ENC_RETURN ENC_Set_Movie_File(UINT enc_id, UINT enc_mode_flag, LPCTSTR enc_file);
```

Description:

Get the movie filename.

Arguments:

<code>enc_id</code>	encoder identifier
<code>enc_mode_flag</code>	encode file type
<code>enc_file</code>	address of the encode file name

Return values:

<code>ENC_SUCCESS</code>	Success
<code>ENC_FAIL</code>	Failure

Note:

Value	Explanation
<code>enc_mode_flag</code>	encode file type
<code>ENC_RECORD_AUDIO</code>	audio file
<code>ENC_RECORD_VIDEO</code>	video (elementary stream) file
<code>ENC_RECORD_PS</code>	Program stream file or system stream file

Reference:

ENC_Get_Frame_Count

ENC_RETURN ENC_Get_Frame_Count(UINT enc_id, UINT* frame_count);

Description:

Get the encoded frame count.

Arguments:

enc_id	encoder identifier
frame_count	address of the frame count

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Get_Time

ENC_RETURN ENC_Get_Time(UINT enc_id, double* time);

Description:

Get the encoded time.

Arguments:

enc_id	encoder identifier
time	address of the encode time (seconds)

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

ENC_Detect_Video_Input_Source

ENC_RETURN ENC_Detect_Video_Input_Source(UINT enc_id, LPINT lpfInputSource);

Description:

Automatically detect the video input source.

Arguments:

enc_id	encoder identifier
lpfInputSource	pointer to the detected video input source

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Value	Explanation
ENC_VIDEO_INPUT_SOURCE_COMPOSITE	Composite
ENC_VIDEO_INPUT_SOURCE_SVIDEO	S-Video

Reference:

ENC_Get_Video_Parameter, ENC_Set_Video_Parameter

ENC_Set_Digital_Video_Input

ENC_RETURN ENC_Set_Digital_Video_Input(UINT enc_id, BOOL fDigital);

Description:

Switch between analog video input and digital video input.

Arguments:

enc_id	encoder identifier
fDigital	value to set the video input

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Value	Explanation
True	Digital Input
False	Analog Input

If you are changing the video input while monitoring, please use **ENC_Stop_Monitor** to stop monitoring and then use **ENC_Set_Digital_Video_Input** to change the source. Use **ENC_Start_Monitor** to start monitoring again.

ENC_Get_Digital_Video_Input

ENC_RETURN ENC_Get_Digital_Video_Input(UINT enc_id, LPINT lpfDigital);

Description:

Get the current video input status.

Arguments:

enc_id	encoder identifier
fDigital	value to set the video input

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Value	Explanation
True	Digital Input
False	Analog Input

ENC_Set_Video_Encode_File

ENC_RETURN ENC_Set_Video_Encode_File(UINT enc_id,LPCTSTR lpszVEFile);

Description:

Set the file name to set the detailed encode parameter when encoding starts.

Arguments:

enc_id	encoder identifier
lpszVEFile	pointer to detailed video encode parameter file

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

If you set **NULL** to **lpszVEFile**, there will be no settings for the detailed video encode parameter. But if you do encoding once and then set **null** to **lpszVEFile**, the previous setting will still be applied. To erase the settings, please terminate the encoder using **ENC_Terminate**.

For details of the video encode parameter file, see **VideoEncoderParams.txt** in the setup CD-ROM.

ENC_Get_Last_Error

ULONG ENC_Get_Last_Error(UINT enc_id);

Description:

Get the last error.

Arguments:

enc_id	encoder identifier
--------	--------------------

Return values:

ULONG	Error code(see appendix 1 .error codes)
-------	---

Note:

When you run **ENC_Get_Last_Error**, the error information is cleared. If there are no new errors, **MVR_ERROR_SUCCESS** is returned.

Video Decoder Functions

DEC_Set_Callback

DEC_RETURN DEC_Set_Callback(UINT dec_id, DEC_CB_STATUS status, DEC_CB_ERROR error);

Description:

Set the callback function.

Arguments:

dec_id	decoder identifier
status	set the internal state notification callback function
error	set the error notification callback function

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC_CB_STATUS

VOID CALLBACK StatusProc(UINT dec_id, DEC_STATUS_NOTIFY status);

Description:

Callback function to get the decoder's internal state.

Arguments:

dec_id	decoder identifier
status	internal state

Return values:

Note:

Status Value	Explanation
DEC_NOTIFY_INITIALIZE	decoder initialized
DEC_NOTIFY_TERMINATE	decoder was terminated
DEC_NOTIFY_STOP	decode stopped
DEC_NOTIFY_PAUSE	decode paused
DEC_NOTIFY_PLAY	decode started

Reference:

DEC_CB_ERROR

VOID CALLBACK ErrorProc(UINT dec_id, ULONG error);

Description:

Callback function to get the decoder's error information.

Arguments:

<code>dec_id</code>	decoder identifier
<code>error</code>	error code (see error code reference in APPENDIX)

Return values:

Note:

The callback function set in **DEC_CB_ERROR** is run when there is an error during encoding. IF **DEC_FAIL** is returned, get the error message using **DEC_Get_Last_Error**.

Reference:

DEC_Get_Last_Error

DEC Can Initialize

DEC_RETURN DEC_Can_Initialize(UINT dec_id);

Description:

Check if the decoder can be initialized.

Arguments:

dec_id	decoder identifier
--------	--------------------

Return values:

DEC_SUCCESS	can be initialized
DEC_FAIL	cannot be initialized

Note:

Reference:

DEC Get Codec Config

```
DEC_RETURN DEC_Get_Codec_Config(UINT dec_id, DEC_CONFIG*pConfig);
```

Description:

Get card information

Arguments:

dec_id	decoder identifier
pConfig	pointer to DEC_CONFIG structure

Note:

If you set the **CodecNumber** in **DEC_CONFIG** to **0**, you can get the number of current cards in the system, and the card that is currently being used. Number of cards will be stored in **NumberCodecs**. The currently used card is stored in **CodecNumber**. If no cards are used, **0** will be stored in **CodecNumber**.

DEC_CONFIG

```
typedef struct{
    UINT    NumberCodecs;
    UINT    CodecNumber;
    UINT    CodecCaps;
    UINT    CurrentUtilization;
}DEC_CONFIG;
```

Description:

get information about the card

Arguments:

NumberCodecs	stores the number of cards installed
CodecNumber	set 0 or the card number of the card you need the information of (1 – NumberCodecs)
CodecCaps	stores the information flag for the card set at CodecNumber

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Status value	Explanation
DECCAP_NONE	no information
DECCAP_ENCODE	flag to show encode is possible
DECCAP_DECODE	flag to show decode is possible
DECCAP_MONITOR	flag to show monitoring is possible
DECCAP_AVI2MPEG	flag to show AVI to MPEG convert is possible

Arguments:

CurrentUtilization stores the info of the card stated at CodecNumber

Note:

Status value	Explanation
TRUE	Used
FALSE	Not used

DEC_Set_Codec_Number

```
DEC_RETURN DEC_Set_Codec_Number(UINT dec_id, UINT nNumber);
```

Description:

Set the card number of the card to be used

Arguments:

dec_id	decoder identifier
nNumber	card number

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Run ***DEC_Set_Codec_Number*** before initializing using ***DEC_Initialize*** or ***DEC_Initialize_Ex***. If you don't run ***DEC_Set_Codec_Number*** before initializing, a card will be selected automatically before initializing.

The card number is from **1** to the value of ***NumberCodecs***, in the ***DEC_CONFIG*** structure in the ***DEC_Get_Codec_Config***.

If the number is set to **0**, a card will be selected automatically.

Reference:

DEC_Get_Codec_Config, ***DEC_Initialize***, ***DEC_Initialize_Ex***

DEC Initialize

DEC_RETURN DEC_Initialize(UINT dec_id);

Description:

Initialize the decoder.

Arguments:

dec_id	decoder identifier
--------	--------------------

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC Initialize Ex

```
DEC_RETURN DEC_Initialize_Ex(UINT dec_id, DEC_MEDIA media,
                             DEC_BSR_PARAMETER* pParam);
```

Description:

Initialize the decoder, and set the media and memory transfer settings.

Arguments:

dec_id	encoder identifier
media	encoder type

Status value

DEC_MEDIA_FILE
DEC_MEDIA_MEMORY

Explanation

decode file
transfer from memory

pParam	The pointer pParam for DEC_BSR_PARAMETER structure, that sets the parameter for memory transfer, is set when media is DEC_MEDIA_MEMORY . Otherwise, NULL is returned.
--------	--

DEC BSR PARAMETER

```
typedef struct {
    DEC_BSR_TYPE    fType;
    DEC_CB_BSR      pfnCallback;
    DWORD           cbBuff;
    DWORD           cBuff;
    DWORD           dwData;
} ENC_BSR_PARAMETER;
```

fType	reserved (returns DEC_BSR_TYPE_PS)
pfnCallback	set callback function for memory transfer
cbBuff	buffer size for memory transfer during callback. Size is increment of pack size (2048 bytes)
cBuff	size of buffer set in cbBuff
dwData	32 bit data sent during callback

Description:

Set the parameter info for memory transfer.

DEC CB BSR

VOID CALLBACK BsrProc(LPBYTE pbBuff, DWORD cbWrite, DWORD pcbWritten, DWORD dwData);

pbBuff	pointer to buffer where the decode data is stored
cbWrite	decode data size
pcbWritten	pointer to buffer where decode data size is stored
dwData	value of dwData set in DEC_BSR_PARAMETER structure

Description:

Callback function for transferring decoded data.

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

DEC_Terminate

DEC_RETURN DEC_Terminate(UINT dec_id);

Description:

Terminate the decoder.

Arguments:

dec_id	decoder identifier
--------	--------------------

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

All callback settings in **DEC_Set_Callback** is terminated after **DEC_Terminate**

Reference:

DEC_Set_Callback

DEC_Get_Media

DEC_RETURN DEC_Get_Media(UINT dec_id, DEC_MEDIA* media);

Description:

Get the decode type.

Arguments:

enc_id	encoder identifier
media	pointer to DEC_MEDIA (Get decode type)

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Reference:

DEC_Initialize_Ex

DEC_Set_Media

DEC_RETURN DEC_Set_Media(UINT dec_id, DEC_MEDIA media);

Description:

Set the decode type.

Arguments:

dec_id	encoder identifier
media	encode type

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Reference:

ENC_Initialize_Ex

DEC Can Playback

DEC_RETURN DEC_Can_Playback(UINT dec_id);

Description:

Checks if playback is possible or not.

Arguments:

dec_id	encoder identifier
--------	--------------------

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Reference:

DEC Get Status

DEC_RETURN DEC_Get_Status(UINT dec_id, DEC_STATUS* status);

Description:

Get the internal state of the decoder.

Arguments:

dec_id	decoder identifier
status	address of the DEC_STATUS structure

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Status Value	Explanation
DEC_STATUS_WAITINITIALIZE	initialization wait state
DEC_STATUS_WAITTERMINATE	terminate wait state
DEC_STATUS_STOP	stopped
DEC_STATUS_PAUSE	paused
DEC_STATUS_PLAY	decoding

Reference:

DEC Can Overlay Window

DEC_RETURN DEC_Can_Overlay_Window(UINT dec_id, HANDLE hMonotior, UINT nWidth, UINT nHeight);

Description:

Check if overlay window can be used.

Arguments:

dec_id	decoder identifier
hMnotior	monitor handle
nWidth	overlay window width
nHeight	overlay window height

Return values:

DEC_SUCCESS	Overlay window can be used
DEC_FAIL	Overlay window can not be used

Note:

In **hMonitor**, set the monitor handle (primary or secondary) in multi monitor systems. If not multi monitor, set **NULL** in **hMonitor**.

Reference:

DEC Create Overlay Window

DEC_RETURN DEC_Create_Overlay_Window(UINT dec_id, HWND hWndParent, HWND* hWndOverlay, int x, int y, UINT nWidth, UINT nHeight);

Description:

Create an overlay window.

Arguments:

dec_id	decoder identifier
hWndParent	parent window handle
hWndOverlay	overlay window handle acquires address
x	overlay window position (left)
y	overlay window position (top)
nWidth	overlay window width (Max. 720)
nHeight	overlay window height (Max. NTSC 480, PAL 576)

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

When creating the overlay window, the maximum size limitations apply. After creating the overlay window, it can be resized above those limitations by using **DEC_Resize_Overlay_Window**. The Overlay window can not be created during decoding.

Reference:

DEC_Resize_Overlay_Window

DEC_Destroy_Overlay_Window

DEC_RETURN DEC_Destroy_Overlay_Window(UINT dec_id);

Description:

Destroy the overlay window.

Arguments:

dec_id	decoder identifier
--------	--------------------

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Cannot interrupt overlay window while decoding.

Reference:

DEC Move Overlay Window

DEC_RETURN DEC_Move_Overlay_Window(UINT dec_id, INT x, INT y);

Description:

Move the overlay window.

Arguments:

<code>dec_id</code>	decoder identifier
<code>x</code>	client region of parent window upper left corner X coordinate
<code>y</code>	client region of parent window upper right corner Y coordinate

Return values:

<code>DEC_SUCCESS</code>	Success
<code>DEC_FAIL</code>	Failure

Note:

Reference:

DEC Resize Overlay Window

DEC_RETURN DEC_Resize_Overlay_Window(UINT dec_id, UINT nWidth, UINT nHeight);

Description:

Resize the overlay window.

Arguments:

dec_id	decoder identifier
nWidth	overlay window width
nHeight	overlay window height

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC Show Overlay Window

DEC_RETURN DEC_Show_Overlay_Window(UINT dec_id, BOOL fShow);

Description:

Change the overlay window display state.

Arguments:

dec_id	decoder identifier
fShow	display the overlay window (TRUE), do not display the overlay window (FALSE)

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC Get Overlay Window

DEC_RETURN DEC_Get_Overlay_Window(UINT dec_id, HWND* pWnd);

Description:

Get the overlay window handle.

Arguments:

dec_id	decoder identifier
pWnd	acquire overlay window handle address

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC_Get_Overlay_Rect

DEC_RETURN DEC_Get_Overlay_Rect(UINT dec_id, LPRECT pRect);

Description:

Get the overlay window area info.

Arguments:

dec_id	decoder identifier
pRect	pointer to RECT structure to get the overlay window area info

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC_Set_Overlay_Rect

```
DEC_RETURN DEC_Set_Overlay_Rect(UINT dec_id, LPRECT pRect);
```

Description:

Set the overlay window area.

Arguments:

dec_id	decoder identifier
pRect	pointer to RECT structure to get the overlay window area info

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

The area that can be set with **pRect** is as below;

Video Format	Horizontal Position	Vertical Position
NTSC	0 to 720	0 to 480
PAL	0 to 720	0 to 576

Depending on the playback screen size, the following calculation will be applied to the value;

Video Format	Horizontal value	Vertical value
NTSC SIF or under	x 2	x 2
NTSC Half-D1 or under	x 2	x 1
NTSC other	x 1	x 1
PAL SIF or under	x 2	x 2
PAL Half-DIF or under	x 2	x 1
PAL other	x 1	x 1

This function can not be set while decoding.

DEC Start Monitor

DEC_RETURN DEC_Start_Monitor(UINT dec_id);

Description:

Start the monitor.

Arguments:

dec_id	decoder identifier
--------	--------------------

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

When using **DEC_Start_Monitor** the overlay window returns to the size at time of creation.
Cannot be started during decoding.

Reference:

DEC Stop Monitor

DEC_RETURN DEC_Stop_Monitor(UINT dec_id);

Description:

Stop the monitor.

Arguments:

dec_id	decoder identifier
--------	--------------------

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Cannot be stopped during decoding.

Reference:

DEC Get Monitor Status

DEC_RETURN DEC_Get_Monitor_Status(UINT dec_id, UINT* monitor);

Description:

Get the monitor status.

Arguments:

dec_id	decoder identifier
monitor	monitor status – TRUE = Monitoring, FALSE = Not monitoring

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC Get BSR Parameter

DEC_RETURN DEC_Get_BSR_Parameter(UINT dec_id, DEC_BSR_PARAMETER*pParam);

Description:

Get the parameter for memory transfer.

Arguments:

dec_id	decoder identifier
pParam	pointer to DEC_BSR_PARAMETER structure (to get memory transfer parameter)

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC_Initialize_Ex

DEC Set BSR Parameter

```
DEC_RETURN DEC_Set_BSR_Parameter(UINT dec_id, DEC_BSR_PARAMETER*pParam);
```

Description:

Set the parameter for memory transfer.

Arguments:

dec_id	decoder identifier
pParam	pointer to DEC_BSR_PARAMETER structure (to set memory transfer parameter)

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC_Initialize_Ex

DEC Get Overlay Parameter

```
DEC_RETURN DEC_Get_Overlay_Parameter(UINT dec_id, DEC_OVERLAY_PARAMETER*pParam);
```

Description:

Get the parameter for displaying overlay.

Arguments:

dec_id	decoder identifier
pParam	pointer to DEC_OVERLAY_PARAMETER structure (to get overlay parameter)

DEC Overlay Parameter

```
typedef struct {  
    UINT nOverlayBrightness;  
    int nOverlayContrast;  
    int nOverlaySaturation;  
} DEC_OVERLAY_PARAMETER;
```

Description:

Stores the parameter for displaying overlay.

nOverlayBrightness	brightness value DECMIN_VIDEO_OVERLAY_BRIGHTNESS = minimum DECMAX_VIDEO_OVERLAY_BRIGHTNESS = Maximum DECDEF_VIDEO_OVERLAY_BRIGHTNESS = default
nOverlayContrast	contrast value DECMIN_VIDEO_OVERLAY_CONTRAST = minimum DECMAX_VIDEO_OVERLAY_CONTRAST = Maximum DECDEF_VIDEO_OVERLAY_CONTRAST = default
nOverlaySaturation	saturation value DECMIN_VIDEO_OVERLAY_SATURATION = minimum DECMAX_VIDEO_OVERLAY_SATURATION = Maximum DECDEF_VIDEO_OVERLAY_SATURATION = default

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

DEC_Set_Overlay_Parameter

DEC_RETURN DEC_Set_Overlay_Parameter(UINT dec_id, DEC_OVERLAY_PARAMETER*pParam);

Description:

Set the parameter for overlay.

Arguments:

dec_id	decoder identifier
pParam	pointer to DEC_OVERLAY_PARAMETER structure (to set overlay parameter)

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC_Get_Overlay_Parameter

DEC Get Video Parameter

DEC_RETURN DEC_Get_Video_Parameter(UINT dec_id, DEC_VIDEO_PARAMETER* Video_Param);

Description:

Get the video parameters.

Arguments:

dec_id	decoder identifier
Video_Param	pointer to a DEC_VIDEO_PARAMETER structure

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

DEC_VIDEO_PARAMETER structure definition

```
typedef struct {  
    int    TV_System;  
} DEC_VIDEO_PARAMETER;
```

Value

TV_System
DEC_VIDEO_TV_SYSTEM_NTSC
DEC_VIDEO_TV_SYSTEM_PAL

Explanation

Broadcast standard settings.
NTSC
PAL

Reference:

DEC_Set_Video_Parameter

DEC_RETURN DEC_Set_Video_Parameter(UINT dec_id, DEC_VIDEO_PARAMETER* Video_Param);

Description:

Set the Video parameter settings.

Arguments:

enc_id	encoder identifier
Video_Param	pointer to a DEC_VIDEO_PARAMETER structure

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

If changing the **TV_System**, **fInputSource**, **fOutputMonitor** values of **DEC_VIDEO_PARAMETER** during monitoring then stop the monitor by using **DEC_Stop_Monitor** before changing the values. To resume use **DEC_Start_Monitor**.

Reference:

DEC_Stop_Monitor, **DEC_Start_Monitor**, **DEC_Get_Video_Parameter**

DEC Get Audio Parameter

```
DEC_RETURN DEC_Get_Audio_Parameter(UINT dec_id, DEC_AUDIO_PARAMETER*
                                   Audio_Param);
```

Description:

Get the audio parameters.

Arguments:

dec_id	decoder identifier
Audio_Param	pointer to DEC_AUDIO_PARAMETER structure

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

DEC_AUDIO_PARAMETER structure definition

```
typedef struct {
    BOOL          fOutputMute;
    DWORD         dwLeftOutputAtt;
    DWORD         dwRightOutputAtt;
} DEC_AUDIO_PARAMETER;
```

Value	Explanation
fOutputMute	mute state
TRUE	mute ON
FALSE	mute OFF
dwLeftOutputAtt	left volume
ENCMIN_AUDIO_OUTPUT_ATT	minimum value
ENCMAx_AUDIO_OUTPUT_ATT	maximum value
dwRightOutputAtt	right volume
ENCMIN_AUDIO_OUTPUT_ATT	minimum value
ENCMAx_AUDIO_OUTPUT_ATT	maximum value

Reference:

DEC Set Audio Parameter

```
DEC_RETURN DEC_Set_Audio_Parameter(UINT dec_id, DEC_AUDIO_PARAMETER*  
                                   Audio_Param);
```

Description:

Set the audio parameters.

Arguments:

dec_id	decoder identifier
Audio_Param	pointer to DEC_AUDIO_PARAMETER structure

Return values:

ENC_SUCCESS	Success
ENC_FAIL	Failure

Note:

Reference:

DEC_Get_Audio_Parameter

DEC Get Decode Parameter

```
DEC_RETURN DEC_Get_Decode_Parameter(UINT dec_id, DEC_DECODE_PARAMETER*
                                     Decode_Param);
```

Description:

Get the decoder parameters.

Arguments:

dec_id	decoder identifier
Decode_Param	pointer to DEC_DECODE_PARAMETER structure

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

DEC_DECODE_PARAMETER structure definition (Parameters may be added in the future)

```
typedef struct {
    BOOL fTimestampUsage;
    BOOL fRepeatOnStop;
    BOOL fFrameRepeat;
    BOOL fPanScan;
    BOOL fFrameClip;
} DEC_DECODE_PARAMETER;
```

Value	Explanation
fTimestampUsage	Use timestamp when playing MPEG1 system stream / MPEG2 program stream
TRUE	use timestamp
FALSE	do not use timestamp
fRepeatOnStop	Repeat last frame after decoder stops
TRUE	display
FALSE	not display
fFrameRepeat	when decode stops, the repeated image
TRUE	frame
FALSE	field
fPanScan	in 16:9 images, decide if there is image area size info, to show it in 16:9
TRUE	Show in 16:9
FALSE	Do not show in 16:9
fFrameClip	set to clip frame edges during playback
TRUE	Clip image during playback
FALSE	Do not clip image during playback

Reference:

DEC_Set_Decode_Parameter

```
DEC_RETURN DEC_Set_Decode_Parameter(UINT dec_id, DEC_DECODE_PARAMETER*
                                     Decode_Param);
```

Description:

Set the decode parameters.

Arguments:

dec_id	decoder identifier
Decode_Param	pointer to DEC_DECODE_PARAMETER structure

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

If **fPanScan** in **DEC_DECODE_PARAMETER** is **TRUE**, the enlarges is as below:

Video Format	Horizontal	Vertical
NTSC Standard	x4/3	x1
NTSC SIF	x8/3	x2
NTSC Half-D1	x8/3	x1
PAL Standard	x4/3	x1
PAL SIF	x8/3	x2
PAL Half-D1	x8/3	x1

If **fPanScan** is **FALSE**, or there is no data about the display area size, the enlarges is as below:

Video Format	Horizontal	Vertical
NTSC Standard	x1	x1
NTSC SIF	x2	x2
NTSC Half-D1	x2	x1
PAL Standard	x1	x1
PAL SIF	x2	x2
PAL Half-D1	x2	x1

In **MPEG1**, **fPanScan** will be ignored.

Reference:

DEC_Get_Decode_Parameter

DEC Play

DEC_RETURN DEC_Play(UINT dec_id);

Description:

Start decoding.

Arguments:

dec_id	decoder identifier
--------	--------------------

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC Play From

DEC_RETURN DEC_Play_From(UINT dec_id,DWORD dwPosition);

Description:

Start decoding from the selected position.

Arguments:

dec_id	decoder identifier
dwPosition	Position to start playback(milliseconds)

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC_Seek

DEC Pause

DEC_RETURN DEC_Pause(UINT dec_id);

Description:

Pause decoding.

Arguments:

dec_id	decoder identifier
--------	--------------------

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC Resume

DEC_RETURN DEC_Resume(UINT dec_id);

Description:

Resume decoding.

Arguments:

dec_id	decoder identifier
--------	--------------------

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC_Stop

DEC_RETURN DEC_Stop(UINT dec_id);

Description:

Stop decoding.

Arguments:

dec_id	decoder identifier
--------	--------------------

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC_Get_Repeat

DEC_RETURN DEC_Get_Repeat(UINT dec_id, UINT* repeat);

Description:

Get the repeat state.

Arguments:

dec_id	decoder identifier
repeat	repeat state

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC_Set_Repeat

DEC_Set_Repeat

DEC_RETURN DEC_Set_Repeat(UINT dec_id, UINT repeat);

Description:

Set the repeat state.

Arguments:

dec_id	decoder identifier
repeat	repeat state

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Value	Explanation
repeat	repeat state
TRUE	repeat ON
FALSE	repeat OFF

Note:

Repeat is valid in file decode only.
In memory transfer, repeat can be set, but not valid.

Reference:

DEC_Initialize_Ex, DEC_Get_Media

DEC Get Movie File

DEC_RETURN DEC_Get_Movie_File(UINT dec_id, LPTSTR dec_file);

Description:

Get the decode file name.

Arguments:

dec_id	decoder identifier
dec_file	address of the decode file name

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC Set Movie File

DEC_RETURN DEC_Set_Movie_File(UINT dec_id, LPTSTR dec_file);

Description:

Set the decode file name.

Arguments:

dec_id	decoder identifier
dec_file	address of the decode file name

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Can not set in memory transfer mode.
After running, decode will start from top.

Reference:

DEC_Initialize_Ex

DEC Get Image Size

DEC_RETURN DEC_Get_Image_Size(UINT dec_id, UINT* image_width, UINT* image_height);

Description:

Get the image size.

Arguments:

dec_id	decoder identifier
image_width	image width
image_height	image height

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC Get Frame Count

DEC_RETURN DEC_Get_Frame_Count(UINT dec_id, UINT* frame_count);

Description:

Get the decoded video stream frame count.

Arguments:

dec_id	decoder identifier
frame_count	address of the frame count

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

When displaying the overlay if the overlay window size or position are changed the frame count will not be accurate.

Reference:

DEC_Get_Time

DEC_RETURN DEC_Get_Time(UINT dec_id, double* time);

Description:

Get the decode time.

Arguments:

dec_id	decoder identifier
time	address of the decode time (seconds)

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

When displaying the overlay, if the overlay window size or position are changed the decode time cannot be updated.

Reference:

DEC_Get_Type

DEC_RETURN DEC_Get_Type(UINT dec_id, DEC_TYPE* type);

Description:

Get the decode type of the decode file.

Arguments:

dec_id	decoder identifier
type	pointer to the DEC_TYPE

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Value type	Explanation
DEC_TYPE_UNKNOWN	decode types unknown
DEC_TYPE_MPEG1_AUDIO	MPEG1 audio
DEC_TYPE_MPEG1_VIDEO	MPEG1 video
DEC_TYPE_MPEG1_SYSTEM_STREAM	MPEG1 system stream
DEC_TYPE_MPEG2_AUDIO	MPEG2 audio
DEC_TYPE_MPEG2_VIDEO	MPEG2 video
DEC_TYPE_MPEG2_PROGRAM_STREAM	MPEG2 program stream

Reference:

DEC_Get_File_Type

DEC_RETURN DEC_Get_File_Type(UINT dec_id, LPCTSTR dec_file, DEC_TYPE* type);

Description:

Get the file type of the decode file.

Arguments:

dec_id	decoder identifier
dec_file	address of the filename
type	pointer to the DEC_TYPE

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Reference:

DEC_Get_Type

DEC Get Playback Time

DEC_RETURN DEC_Get_Playback_Time(UINT dec_id, LPDWORD lpdwTime);

Description:

Get the playback time.

Arguments:

dec_id	decoder identifier
lpdwTime	pointer to the buffer that gives the playback time (in milliseconds)

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Get the playback time of the file set at DEC_Set_Movie_File.

You can get the playback time from files that can adjust playback positions by using DEC_Seek.

Reference:

DEC_Set_Movie, DEC_Seek

DEC Seek

DEC_RETURN DEC_Seek(UINT dec_id, DWORD dwPosition);

Description:

Set the playback time position.

Arguments:

dec_id	decoder identifier
dwPosition	the position where playback starts (in milliseconds)

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

Limitations:

1. The following files do not support this function because in program streams and video elementary streams, this will use the timecode in the GOP.
 - Files with no GOP
 - Files with no timecode
 - Files longer than 24 hours (Time code goes up to only 24 hours)
2. You cannot jump to locations shorter than a GOP (usually 0.4 to 1.0 seconds)
3. VideoCD files (.dat files) are not supported

Playback start positions:

1. In program streams/video elementary streams, it will get the top frame of the closest GOP, without passing the selected timecode.
2. In audio elementary streams, it will get the top of the audio access unit without passing the selected timecode.

DEC_Get_Sync_Stc_Value

DEC_RETURN DEC_Get_Sync_Stc_Value(UINT dec_id, LPDWORD pdwValue);

Description:

Get the SYNC STC resistor value in HD814210

Arguments:

dec_id	decoder identifier
pdwValue	pointer to the variable to get the SYNC STC resistor value of HD814210

Return values:

DEC_SUCCESS	Success
DEC_FAIL	Failure

Note:

The handling when reading from the decoder core resistor is done internally in **MvrAvc.dll**, so to call it you only need to set the decoder core resistor.

This API is used only in the transmission sample program.

Limitations:

DEC_Get_Last_Error

ULONG DEC_Get_Last_Error(UINT dec_id);

Description:

Get the last error.

Arguments:

dec_id	decoder identifier
--------	--------------------

Return values:

ULONG	Error code(see appendix 1 .error codes)
-------	---

Note:

When you run **DEC_Get_Last_Error**, the error information is cleared. If there are no new errors, **MVR_ERROR_SUCCESS** is returned.

APPENDIX

Error Code summary

Value	Meaning
MVR_ERROR_SUCCESS	normal close
MVR_ERROR_PENDING	specified operation is not complete
MVR_ERROR_SYSTEM_ERROR	system error
MVR_ERROR_NOT_PENDING	no specified operations are incomplete
MVR_ERROR_INVALID_ASYNCOP	invalid event handle
MVR_ERROR_UNSUPPORTED	unsupported function was executed
MVR_ERROR_INVALID_TYPE	session type invalid
MVR_ERROR_INVALID_OPTION	session option invalid
MVR_ERROR_INVALID_ADDRESS	invalid address
MVR_ERROR_LIMIT_EXCEEDED	limit value exceeded
MVR_ERROR_INVALID_HANDLE	invalid handle
MVR_ERROR_INVALID_STREAM	invalid stream
MVR_ERROR_INVALID_BUFFER	invalid buffer
MVR_ERROR_INVALID_MESSAGE	invalid message
MVR_ERROR_CANCELED	specified request cancelled
MVR_ERROR_BUSY	specified handle in use
MVR_ERROR_BUFFER_ERROR	buffer initial failure
MVR_ERROR_MESSAGE_ERROR	message error
MVR_ERROR_BAD_PARAM	invalid parameter specified
MVR_ERROR_HARDWARE_ERROR	hardware error
MVR_ERROR_ALLOC_FAILURE	resource allocation failure
MVR_ERROR_INVALID_EVENT	invalid event
MVR_ERROR_INVALID_STATE	invalid status
MVR_ERROR_CURRENT_STATE	Stream is in previously specified state
MVR_ERROR_INVALID_FRAME_TYPE	invalid frame type
MVR_ERROR_INSUFFICIENT_BUFFERS	insufficient buffer size
MVR_ERROR_TIMEOUT	timeout started
MVR_ERROR_INVALID_CODEC_STATE	invalid codec state
MVR_ERROR_INVALID_FORMAT	invalid stream format
MVR_ERROR_INVALID_OPERATION	specified operation invalid
MVR_ERROR_OVERFLOW	overflow created
MVR_ERROR_UNDERFLOW	underflow created
MVR_ERROR_SESSION_CANCELED	session cancelled
MVR_ERROR_UCODE_NOT_FOUND	microcode cannot be found
MVR_ERROR_AUDIO_UCODE_NOT_FOUND	audio microcode cannot be found
MVR_ERROR_PARSE_ERROR	invalid bit stream format
MVR_ERROR_END_OF_STREAM	end of bit stream reached during seek
MVR_ERROR_DISK_FULL	disk full
MVR_ERROR_STREAMS_CLOSED	stream broken
MVR_ERROR_INTERNAL_ERROR	internal error
MVR_ERROR_INVALID_SHORT_BUFFER	buffer not filled perfectly
MVR_ERROR_NO_VIDEO_SOURCE	no video source signal
MVR_ERROR_DEVICE_NOT_FOUND	device not found

MVR_ERROR_EVENT_DATA_OVERRUN

MPEG video data or PCM audio data is not coming from the device fast enough.

(Reasons)

1. PC performance is too slow.
2. Started an another application while capturing, and the data receiving did not catch up.
3. The video signal is invalid : The multiplex program in the driver allocates a buffer for receiving data. If the signal is not correct, or a field is missing in the data, this may result to underrun or overrun in the data transfer from the buffer. This may come from (a) the video tape has degraded and the signal is not coming in at a proper frame rate (NTSC – 29.97Hz), (b) the video tape has been copied over, and irregular signals are in where the scene changes, or (c) the input signal has been cut off.

MVR_ERROR_EVENT_DATA_UNDERRUN

under run created

MVR_ERROR_EVENT_DEVICE_ERROR

device error created

MVR_ERROR_EVENT_LOSS_OF_VIDEO

some input video data missing

MVR_ERROR_EVENT_EVENT_LOSS

event error created

MVR_ERROR_EVENT_UNRECOVERABLE_ERROR

unrecoverable error created (restart PC)

MVR_ERROR_EVENT_WARNING

warning error occurred

MVR_ERROR_EVENT_VIDEO_ENCODER_VBV_UNDERFLOW

failed in bit rate control

MVR_ERROR_EVENT_VIDEO_ENCODER_OVERRUN_STATE

Some frames could not be encoded in time.

(Reasons)

1. PC performance is too slow.
2. Started an another application while capturing.
3. Other I/O (other PCI devices) is taking time.
4. The driver module has been swapped out.