

MICROVIEW VER5.0

SDK程序员手册



公司电话：010—82600088

公司网址：www.microview.com.cn

北京微视新纪元科技有限公司 2009 年 1 月

版权

© 2006 北京微视新纪元科技有限公司 版权所有

第3版，2009年1月

注意：本手册的内容将会变动，且不另行通知。更改的内容将会自动添加到新的出版物当中去。

对本手册内容以外的操作本公司不做任何明示或默许担保。

本公司依据中华人民共和国著作权法，享有及保留本手册的一切著作权，未经本公司的书面许可，不得随意增删、改编、复制及模仿本公司著作。

一旦使用本说明书所列之产品，表示你已经阅读并接受了最终用户许可协议（见附录）中的所有条款。

一般约定：“Microview®”、“微视®”为本公司的商标。

北京微视新纪元科技有限公司

网站：<http://www.microview.com.cn>

销售信箱：sales@microview.com.cn

技术支持信箱：support@microview.com.cn

电话：010-82600088（中继线）

传真：010-82600088-6600

地址：北京市海淀区中关村东路18号财智国际大厦A座10层（100083）

前 言

首先感谢您选用微视图像产品！图像采集卡通用标准接口是我公司提供的应用程序接口函数，它基本上概括了图像卡应用程序编程中涉及的各个方面。通过应用接口库，用户可以很好地了解图像卡的运行机制，并在此基础上进行应用程序的开发。

图像采集卡通用标准接口按照功能可以分为初始化和结束，图像卡的操作、内存操作、存储图像、错误提示等函数。

本手册按功能对接口库中的函数进行了划分，对于每一个功能部分的函数都具体给出了它们的函数原型、参数定义、说明、示例等信息。

本手册为用户进行图像卡的二次开发提供了详细的说明，如果用户在使用过程中发现错误和纰漏，请与我公司的技术支持部门联系，以便我们及时改进。

目 录

| | |
|-----------------------------|-----------|
| 第一部分 程序员指南 | 2 |
| 1.1 概述..... | 2 |
| 1.2 典型的程序流程..... | 2 |
| 1.3 自定义常量 | 4 |
| 1.3.1 自定义结构类型..... | 4 |
| 1.3.2 自定义枚举类型..... | 6 |
| 1.4 函数说明..... | 11 |
| 1.4.1 初始化和结束函数..... | 11 |
| 1.4.2 板卡操作函数..... | 12 |
| 1.4.3 内存操作函数..... | 16 |
| 1.4.4 捕获操作函数..... | 19 |
| 1.4.5 存储图像文件函数..... | 21 |
| 1.4.6 16 位图像转 24 位图像函数..... | 23 |
| 1.4.7 场扩帧函数..... | 24 |
| 1.4.8 错误提示函数..... | 25 |
| 1.4.9 OSD操作函数..... | 26 |
| 1.4.10 外触发操作函数..... | 29 |
| 1.4.11 获得图像数据..... | 31 |
| 1.4.12 自动帧测行场频..... | 33 |
| 1.4.13 RGB分量显示..... | 34 |
| 1.4.14 自定义函数类型..... | 36 |
| 第二部分 配置文件说明 | 44 |
| 2.1 MV_MX0. INI | 44 |
| 2.2 MV7146. INI | 47 |
| 2.3 MV7130. INI | 50 |
| 2.4 MVBt. INI | 53 |
| 2.5 MVSQ. INI | 56 |
| 2.6 MVVGA. INI | 58 |

第一部分 程序员指南

1.1 概述

通用标准接口是我公司在原有mvpci函数结构基础上,参考国内外同行sdk结构中较为优秀合理的部分,同时考虑到客户在实际应用中的不同需求,经过一年多的时间开发出来的新的sdk体系结构。

新的接口分为三大类:

- 1、SDK编程:调用我公司自定义形式函数,实现图像采集、存储、控制等功能。这种接口采用统一的函数命名方式,不同的设备(板卡)使用相同的函数名,消除客户在因硬件更新所造成的软件重新开发的负担;
- 2、TWAIN接口:遵循twain组织的标准接口,实现图像采集,存储,适用于使用标准TWAIN接口开发应用程序和使用基于TWAIN接口的第三方软件的客户,使客户可以直接使用我们的设备(板卡);
- 3、VFW接口:遵循微软的标准视频采集存储接口,可以使用通用的视频采集软件配合软件压缩/解压缩编码器实现图像的软件压缩序列存储。

通用标准接口支持 32 位编程开发工具 Microsoft Visual C/C++、Microsoft Visual Basic、Borland C++ Builder、Delphi 等。

在 Microsoft windows 环境中使用动态库 MVAPI.dll 控制采集卡,库中的所有函数原形均在头文件 MVAPI.h 中声明,自定义常量在头文件 COMMON.h 中声明。另外,COMMON.h 中还包含了 MVBOARD1.h 和 MVBOARD1.h,它们定义了本通用标准接口适用的板卡的类型。

用户进行二次开发,如果使用 C/C++编程工具,用户应在程序中调用相关的包含文件(.h),并将静态链接库(.lib)文件加入到工程文件中,供编译程序在链接时使用。请在自己的程序中加入:

#include "MVAPI.h"及#include "COMMON.h"。

如果使用 MS-Visual Basic、Delphi 等编程工具调用通用标准接口时，应按照调用动态链接库的方法，在程序中重新声明函数原型，这时要注意正确定义参数的数据类型。

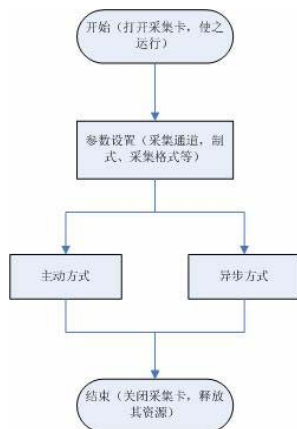
注意：1.本手册中的所有例子均使用 C++(C)语言。

- 2.由于标准的 vfw 和 directshow 的规范中不支持水平、垂直偏移等视频参数的调节功能，为实现这些功能，我们在自己编写的 vfw 接口驱动中通过自定义的话框来对各种视频参数进行调节。我们的 vfw 接口不是完全符合 windows 的标准，因此在 win98 下有可能无法正常使用。所以建议在 windows98 下使用 vfw 的客户可以通过将系统软件升级到 windows2000 或 XP 的方式来使用板卡的 vfw 接口。我们在 windows2000 和 windows XP 上的测试结果是可以正常使用的。

1.2 典型的程序流程

在 Microsoft Windows 操作系统中，图像卡通过标准接口直接操作相机，提供给用户一个简单明确的应用接口。用户在编制自己的应用程序时，可以直接调用这些库函数来实现指定的功能。

图像卡工作流程如下图：



1. 用户只使用采集和显示，主动方式中突出用MV_CaptureSingle来捕获一副图像。

第一步：MV_OpenDevice得到设备句柄；

第二步：MV_OperateDevice带MVRUN参数使设备处于运行状态；

第三步：MV_SetDeviceParameter的第二个参数为DISP_WHND，设置图像显示的窗口句柄；

第四步：现在您可用其他sdk提供的函数功能来调节设备，完成您的工作。如下：

调节设备参数的主要用：

MV_SetDeviceParameter; MV_GetDeviceParameter

操纵设备进入状态的主要用：MV_OperateDevice

输入/输出触发控制的主要用：

MV_SetOutputState; MV_GetInputState;

MV_SetInputCallBack

主动捕获单帧的主要用：

MV_CaptureSingle

捕获单帧后将这一帧存成文件的的主要用：

MV_SaveFile

最后一步：调用MV_CloseDevice来关闭设备。

2. 用户使用采集和显示，异步方式中突出用回调来捕获图像。

第一步：MV_OpenDevice得到设备句柄；

第二步：MV_OperateDevice带MVRUN参数来使设备处于运行状态；

第三步：MV_SetDeviceParameter带DISP_WHND参数为显示设置一个窗口句柄，该步和第四步次序可互换；

第四步：用MV_SetCallBack来为捕获设置回调，当每一场/帧采集结束时该回调被调用并且携带数据和参数。

... ..

最后一步：调用MV_CloseDevice来关闭设备。

一般情况下，图像卡的开始操作和初始化参数的设置，最好在用户应用程序的初始化中完成，图像卡的结束操作应在应用程序退出前执行。

1.3 自定义常量

1.3.1 自定义结构类型

```
1. typedef struct _MV_IMAGEINFO_{
    ULONG Length;    // 图像的大小，以字节计
    ULONG nColor;    // 图像的颜色
    ULONG Height;    // 图像的高度
    ULONG Width;     // 图像的宽度
    ULONG SkipPixel; // 每行跳过的像素
}MV_IMAGEINFO, *PMV_IMAGEINFO;
```

图像的描述结构，描述图像的属性。

应注意SkipPixel成员，它表明了每行在起始时应该丢弃的像素个数。这时候有效的宽度为（图像的宽 - 每行跳过的像素）。个别设备的该成员不为0，大多设备为0。

```
2. typedef struct _MV_RGB_{
    HWND R_hWnd;    RECT R_Rect;
    HWND G_hWnd;    RECT G_Rect;
    HWND B_hWnd;    RECT B_Rect;
}MV_RGB, *PMV_RGB;
```

R, G, B三个分量在分离显示时，分别对应各个分量窗口的句柄及其在窗口中的位置大小。

（参见RGB分量显示**MV_SplitRGB**）

```
3. typedef struct GDIOPERATION{  
    PVOID pIs;    // 系统使用，用户无意义  
    GDIOPERATIONFUNC SetGDIText;  
    GDIOPERATIONFUNC SetGDITextColor;  
    GDIOPERATIONFUNC SetGDITextFormat;  
    GDIOPERATIONFUNC SetGDITextPosition;  
    GDIOPERATIONFUNC SetGDIGraph;  
    GDIOPERATIONFUNC SetGDIGraphPen;  
    GDIOPERATIONFUNC SetGDICanCalAll;  
    GDIOPERATIONFUNC SetGDICanCalOne;  
}GDIOPERATION, *PGDIOPERATION
```

GDI图形体操作结构，有设置输出字符串及其颜色，字体格式，显示位置；设置图形及其画笔；设置按类擦除图形以及擦除所有图形。选择不同的操作，pVal1, pVal2, pVal3的设置不同，具体参见自定义函数类型GDIOPERATIONFUNC。

1.3.2 自定义枚举类型

```
1. enum RUNOPER{
    MVSTOP=0,      // 停止工作
    MVRUN,         // 开始工作
    MVPAUSE,       // 暂停工作
    MVQUERYSTATU,  // 查询当前状态
    MVERROR        // 错误的状态
};
```

设备的运行状态的操作枚举。

```
2. enum CALLBACKTYPE{ BEFORE_PROCESS, AFTER_PROCESS, NO_USED };
```

设备采集用户回调函数的使用类型枚举。

BEFORE_PROCESS: 代表回调函数在系统处理前被调用，且回调中的数据为原始的捕获数据。该回调函数在每一帧到达时都会被准确的被调用；

AFTER_PROCESS : 代表回调函数在系统处理后但在显示前被调用，且回调中的数据为系统处理后的数据。该类型回调函数在每一帧到达时不会被准确的调用。

```
3. enum MV_FILETYPE{ RAW=0, BMP, JPEG };
```

存储文件类型枚举。

RAW : 数据存成原始数据文件;
 BMP : 数据存成BMP格式文件;
 JPEG: 数据存成JPEG格式文件

```
4. enum VIDEOSIGNAL{ GLLEVEL = 0, DIVIDER, ISINTERLACE, XSHIFT,
    YSHIFT, XSIZE, YSIZE, VCOR, VCOG, LP,
    LINEFREQUENCY, LINETOTAL, LINEACTIVETIME,
    LINESYNTIME, LINESHOULDER, FRAMENUM,
    FIELDSYNTIME, FIELDTOTAL, FIELDSHOULDER,
    SOURFREQ, FREQSUB };
```

视频信号参数枚举。高级用户使用，一般不用。

```

5. enum MV_PARAMTER{
    GET_BOARD_TYPE=0,           //得到当前设备的类型
    GET_GRAPHICAL_INTERFACE,    //得到图形接口指针
    SET_GARBIMAGEINFO,          //得到当前被采集图像的信息
    SET_DISPIMAGEINFO,          //得到当前被显示图像的信息
    BUFFERTYPE,                 // 控制设备使用的缓存类型
    DEFAULT_PARAM,              // 恢复或存储设备参数
    // 控制显示
    DISP_PRESENCE,              DISP_WHND,          DISP_TOP,          DISP_LEFT,
    DISP_HEIGHT,                DISP_WIDTH,
    // 控制A/D的颜色,通道等调节参数
    ADJUST_STANDARD,            ADJUST_SOURCE, ADJUST_CHANNEL,
    ADJUST_LUMINANCE,            ADJUST_CHROMINANE, ADJUST_SATURATION,
    ADJUST_HUE,                  ADJUST_CONTRAST, ADJUST_R_LUM,
    ADJUST_G_LUM, ADJUST_B_LUM, ADJUST_R_COARSE,
    ADJUST_G_COARSE, ADJUST_B_COARSE,
    // 控制设备的捕获参数
    GARB_XOFF (GRAB_XOFF), GARB_YOFF(GRAB_YOFF), GARB_HEIGHT
    (GRAB_HEIGHT), GARB_WIDTH (GRAB_WIDTH),
    GARB_IN_HEIGHT (GRAB_IN_HEIGHT),
    GARB_IN_WIDTH (GRAB_IN_WIDTH),
    GARB_BITDESCRIBE (GRAB_BITDESCRIBE),
    GARB_WHOLEWIDTH (GRAB_WHOLEWIDTH),
    // 控制设备的工作参数: 顶底倒置, 水平镜像等功能。
    WORK_UPDOWN,                WORK_FLIP,      WORK_SKIP, WORK_SYNC,
    WORK_INTERLACE,             WORK_ISBLACK, WORK_FIELD,
    OSD_MODE // 控制OSD模式(突出、凹下或不叠加)
    //支持V500及Moka-C50系列卡
    TENBIT_MODE,                OUTPUT_VIDEO,
    FILERSELECT1, FILERSELECT2,
    //支持MVBOARD2.h中所有类型的卡
    ADJUST_BACKCOLORKEY,        DISP_FLIP,
    IMAGE_PROCESS,              VIDEO_SINGLE
};

```

该枚举类型用于指定设备和SDK的工作参数。

每一个枚举成员都有适用的设备类型和对应的取值，还应该注意枚举成员在MV_SetDeviceParameter函数或MV_GetDeviceParameter函数中是否有效。如果某个成员规定不可使用在MV_SetDeviceParameter或MV_GetDeviceParameter中，而用户使用了，也不用担心，没有副作用。

MV_PARAMTER中各个枚举量的含义及取值如下：

GET_BOARD_TYPE: 得到当前设备的类型，只在MV_GetDeviceParameter 时有效。

GET_GRAPHICAL_INTERFACE: 得到图形接口指针，即指向GDIOPERATION 结构的指针，可用该图形接口在图像上叠加图形体。为V3系列卡有效。

SET_GARBIMAGEINFO: 得到当前被采集图像的信息，用MV_SetDeviceParameter的Val参数为MV_IMAGEINFO结构的指针。当MV_SetDeviceParameter返回时该结构被以当前的捕获设置填充。

SET_DISPIMAGEINFO: 得到当前被显示图像的信息，用MV_SetDeviceParameter的Val参数为MV_IMAGEINFO结构的指针。当MV_SetDeviceParameter返回时该结构被以当前的捕获设置填充。该信息是原始图像数据被处理后的用于显示的图像的信息。

BUFFERTYPE: 返回或设置当前的缓冲方式。

DEFAULT_PARAM: 不做说明，用户不使用。

DISP_PRESENCE: 得到或设置当前的显示状态。

DISP_WHND: 得到或设置当前的显示窗口的窗口句柄。

DISP_TOP, DISP_LEFT: 得到或设置当前显示窗口的起始位置。

DISP_HEIGHT, DISP_WIDTH: 得到或设置当前显示窗口的大小。

ADJUST_STANDARD: 得到或设置当前捕获的信号标准。

ADJUST_SOURCE: 得到或设置当前信号的来源。

ADJUST_CHANNEL: 得到或设置当前信号的来源的通道。

ADJUST_LUMINANCE, ADJUST_CHROMINANE, ADJUST_SATURATION, ADJUST_HUE, ADJUST_CONTRAST: 得到或设置当前的亮度，色度，饱和度，色调，对比度，仅对V3系列卡有效，值为0 - 255。

ADJUST_R_LUM, ADJUST_G_LUM, ADJUST_B_LUM: 得到或设置当前信号分别在红路分量，绿路分量，蓝路分量上的亮度，仅对LEVIN系列卡有效。值为0 - 255。

ADJUST_R_COARSE, ADJUST_G_COARSE, ADJUST_B_COARSE: 得到或设置

当前信号分别在红路分量，绿路分量，蓝路分量上的对比度，仅对LEVIN系列卡有效。值为0 - 127。

GARB_XOFF (GRAB_XOFF) , **GARB_YOFF (GRAB_YOFF)** : 得到或设置当前视频信号的起始位置。

GARB_HEIGHT (GRAB_HEIGHT) , **GARB_WIDTH (GRAB_WIDTH)** : 得到或设置当前视频信号的大小尺寸。

GARB_IN_HEIGHT (GRAB_IN_HEIGHT) , **GARB_IN_WIDTH (GRAB_IN_WIDTH)** : 得到或设置当前视频信号的输入大小尺寸。仅对V3系列卡有效。

GARB_WHOLEWIDTH (GRAB_WHOLEWIDTH) : 得到或设置当前视频信号的行总数。仅对M10/20, RGB10/20系列卡有效。

GARB_BITDESCRIBE (GRAB_BITDESCRIBE) : 得到或设置当前采集的像素位数或格式。M10/20卡只能为8位DATA_MONOCHOY8, RGB10/20卡为32位DATA_aRGB8888; V3系列卡为全部。

WORK_UPDOWN: 得到或设置图像的上下翻转。

WORK_FLIP: 得到或设置图像的左右翻转。

WORK_SKIP: 得到或设置图像在采集时写入内存的方式。仅对M10 /20, RGB10/20系列卡有效。

WORK_SYNC: 得到或设置同步信号的来源。仅对M10/20, RGB10/20系列卡有效。

WORK_INTERLACE: 得到或设置信号是逐行还是隔行。仅对M10/20, RGB10/20系列卡有效。

WORK_ISBLACK: 得到或设置采集时的灰阶方式。仅对M10/20, RGB10/20系列卡有效。

WORK_FIELD: 得到或设置采集时是按场采集，还是按帧采集。

OSD_MODE: 确定OSD的位屏蔽模式。仅对V3系列卡有效。

TENBIT_MODE: 用于选择10bit模式。仅对V500系列及Moka-C50系列卡有效。

OUTPUT_VIDEO: 用于选择输出复合视频信号的通道号。仅对V500系列及Moka-C50系列卡有效。

FILERSELECT1: 设置陷波滤波器，取值范围0—7。

FILERSELECT2: 设置多频色度滤波器，取值范围0—3。

ADJUST_BACKCOLORKEY: 设置图像在显示时的窗口背景颜色。支持MVBOARD2. h中所有卡。

DISP_FLIP: 图像显示时是否上下翻转。支持MVBOARD2. h中所有卡。

IMAGE_PROCESS: 设置按场采集时是否进行图像处理。支持MVBOARD2. h中所有卡。

VIDEO_SINGLE: 判断是否有视频源信号。支持MVBOARD2. h中所有卡。]

1.4 函数说明

1.4.1 初始化和结束函数

1. MV_OpenDevice

原型: MVAPI HANDLE WINAPI

MV_OpenDevice(DWORD Index, BOOL bRelease)

说明: 初始化, 创建 Index 指定的设备。

入口参数:

Index : 采集卡的序列下标, 是从0开始的整数。

bRelease: 为 TRUE 时使用 Release 版本 SDK 包, 否则使用 DEBUG 版本 SDK 包。当用户在调试程序时设为 FALSE, 当用户希望 Release 自己的应用程序时为设 TRUE, 以提高程序的响应速度。

返回值:

返回Index对应采集卡的设备句柄, 对采集卡的操作均通过该句柄实现。如果返回NULL, 可调用MV_GetLastError得到错误原因。

2. MV_CloseDevice

原型: MVAPI VOID WINAPI MV_CloseDevice(HANDLE hDevice)

说明: 不再使用以 hDevice 标识的设备时, 关闭该设备。

入口参数:

hDevice: 为先前用 MV_OpenDevice 创建并打开的设备句柄。

返回值: 无。

例: (使用多卡时) 假如HANDLE hDev为先前打开的设备句柄, 现在关闭它并打开第二个设备, 使用调试版本。

```
if( hDev != NULL )
    MV_CloseDevice( hDev );
hDev = NULL;
hDev = MV_OpenDevice( 1, FALSE );
if( hDev == NULL )
{
    MV_GetLastError( TRUE );
    return ERROR;
} // OK! 现在hDev为第二个设备句柄
```


1. 4. 2 板卡操作函数

1. MV_GetDeviceNumber

原型: MVAPI DWORD WINAPI MV_GetDeviceNumber()

说明: 返回主机中正确安装的采集卡数目。

入口参数: 无。

返回值: 主机中正确安装的采集卡数目。

2. MV_OperateDevice

原型: MVAPI RUNOPER WINAPI MV_OperateDevice
(HANDLE hDevice, RUNOPER Oper)

说明: 操纵设备, 即使采集卡处于运行/停止/暂停状态。

入口参数: hDevice: 设备句柄。

Oper: 设备的状态(运行, 停止, 暂停, 查询)。

返回值: 返回卡的当前工作状态。当为 MVERRORE 时代表操作失败,
调用 MV_GetLastError 得到错误原因。

例 1: 打开第三个设备使它处于运行状态:

```
HANDLE hDev = MV_OpenDevice( 2, FALSE );
if( hDev == NULL )
{
    MV_GetLastError( TRUE );
    return ERROR;
}
// 为了在一个窗口显示, hWindowHandle为显示窗口的窗口句柄
MV_SetDeviceParameter( hDevice, DISP_WHND, (DWORD) hWindowHandle );
If (MV_OperateDevice(hDevice, MVRUN) == MVERRORE )
```

.....

例 2: 查询卡的工作状态:

```
RUNOPER WorkStatu = MV_OperateDevice ( hDevice, MVQUERYSTATU );
switch( WorkStatu )
{
case MVSTOP : MessageBox( 0, "停止", "当前工作状态", 0 );
              break;
```

```

case MVRUN           : MessageBox( 0, "运行", "当前工作状态", 0 );
                      break;
case MVPAUSE         : MessageBox( 0, "暂停", "当前工作状态", 0 );
                      break;
case MVERERROR       : MessageBox( 0, "错误", "当前工作状态", 0 );
                      break;
}

```

3. MV_SetDeviceParameter

原型: MVAPI BOOL WINAPI MV_SetDeviceParameter(HANDLE
hDevice, MV_PARAMTER Oper, DWORD Val);

说明: 设置设备和 SDK 的工作参数。

入口参数: hDevice: 设备句柄;

Oper: 欲设置工作参数的类型;

Val: 欲设置参数的值。

返回值: 返回 TRUE 代表参数设置成功; FALSE 代表失败, 一般原因为该类型的卡不支持该类型的工作参数。

注意: 每个参数都有适用的卡型和默认的值, 如果你没有设置, 系统将会使用默认值。

例 1: 希望采集 768*576 的图像, 采集格式 RGB32 位 (即 DATA_aRGB8888), 显示为 352*288 且使用 Window GDI 方式显示。

```

// 设置采集为 768*576*RGB32 (即 DATA_aRGB8888)
MV_SetDeviceParameter( hDevice, GARB_WIDTH, 768 );
MV_SetDeviceParameter( hDevice, GARB_HEIGHT, 576 );
MV_SetDeviceParameter( hDevice, GARB_BITDESCRIBE,
                      DATA_aRGB8888 );

// 设置显示为 352*288
MV_SetDeviceParameter( hDevice, DISP_WIDTH, 352 );
MV_SetDeviceParameter( hDevice, DISP_HEIGHT, 288 );
MV_SetDeviceParameter( hDevice, BUFFERTYPE,
                      SYSTEM_MEMORY_GDI );

```

例 2: 得到原始图像数据的描述和处理后图像数据的描述。

```
MV_IMAGEINFO  RawInfo;      //原始图像数据的描述
MV_IMAGEINFO  ProcessedInfo; //处理后图像数据的描述
MV_SetDeviceParameter( hDevice, SET_GARBIMAGEINFO,
                      (DWORD>(&RawInfo) ));
MV_SetDeviceParameter( hDevice, SET_DISPIIMAGEINFO,
                      (DWORD>(&ProcessedInfo) ));
```

注意：无论是处理前还是处理后，都需要注意 MV_IMAGEINFO 结构中 SkipPixel 成员的值，该成员为每行在起始时应跳过的像素个数，要按实际应用根据 SkipPixel 的值对图像的宽进行一些调整。这个例子比较特殊，但十分有用。

4. MV_GetDeviceParameter

原型: MVAPI LONG WINAPI MV_GetDeviceParameter(HANDLE hDevice, MV_PARAMTER Oper)

说明: 得到设备参数的当前值。

入口参数: hDevice：设备句柄。

Oper：欲设置工作参数的类型。

返回值: 返回-1 代表失败，一般原因为该类型的卡不支持该类型的工作参数。否则代表 Oper 参数的当前值。

例: 在当前的亮度的基础上将亮度进行调节

```
// 得到当前的亮度
ULONG  Luminance = MV_GetDeviceParameter( hDev, ADJUST_LUMINANCE );
// 改变当前的亮度
MV_SetDeviceParameter( hDevice, ADJUST_LUMINANCE, ( Luminance + 10) );
```

5. MV_SaveDeviceParam

原型: MVAPI BOOL WINAPI MV_SaveDeviceParam(HANDLE hDevice)

说明: 保存当前工作参数，作为以后的工作参数的默认值。

入口参数: hDevice: 设备句柄。

返回值: 返回 TRUE 调用成功，返回 FALSE 调用失败。调用 MV_GetLastError 得到错误原因。

6. MV_ResetDeviceParam

原型: MVOKE BOOL WINAPI MV_ResetDeviceParam(HANDLE
hDevice)

说明: 将当前工作参数恢复到默认值。

入口参数: 参数 hDevice: 设备句柄。

返回值: 返回 TRUE 调用成功, 返回 FALSE 调用失败。调用
MV_GetLastError 得到错误原因。

1.4.3 内存操作函数

内存操作中的连续采集是一种异步的操作，用户调用启动采集函数后会立刻启动采集，并且立刻返回；当采集的条件满足时 SDK 回给你通知以及用户想要的图像；而且用户也可以在采集时打断采集。

1. MV_AllocSequenceFrameMemory

原型：MVAPI BOOL WINAPI

MV_AllocSequenceFrameMemory (HANDLE hDevice,
ULONG Action, DWORD Number, CALLBACKTYPE
MemoryType)

说明：对设备分配大帧存，调用时指定此帧存用于存放原始的数据或处理后的数据。因为原始数据中图像每个像素所占的位数可能与处理过的数据不同，而且图像的宽度也可能不同。

入口参数：hDevice : 设备句柄。

Number : 欲分配帧存能存放的图像的数目。

Action : 指定帧存的类型，保留待扩展。可为用户定制大帧存操控的方式。

MemoryType: 指定存放原始数据还是处理后的数据，即 BEFORE_PROCESS 或 AFTER_PROCESS。

返回值：返回 TRUE 分配成功；返回 FALSE 分配失败。

该函数可多次使用，用户无需担心会有内存泄漏。但是应该注意的是当您分配了很大的帧存却长期不使用时，应调用 MV_FreeSequenceFrameMemory 来释放该内存，否则会对操作系统的内存管理造成很大的压力。

2. MV_FreeSequenceFrameMemory

原型：MVAPI BOOL WINAPI MV_FreeSequenceFrameMemory (HANDLE hDevice)

说明：释放设备先前分配的大帧存。一般来说分配和释放是一一对应的，但这里用户编程时可以多次调用分配帧存函数，而在最后应用程序退出时调用释放函数。

入口参数：参数 hDevice: 设备句柄。

返回值：返回TRUE释放成功；返回FALSE释放失败。

3. MV_SequenceCaptureStart

原型: MVAPI BOOL WINAPI MV_SequenceCaptureStart (HANDLE
hDevice, CONTINUEGARBMECHANISM pContinueCall, PVOID
pContext)

说明: 开始进行连续捕获, 捕获到的图像存放在用户分配的帧存中。
当连续帧捕获结束或用户调用MV_SequenceCaptureStop函数
发出停止捕获命令, 用户设置的回调将会被调用。

入口参数: hDevice : 设备句柄。
pContinueCall: 连续捕捉机制的用户回调, 捕获结束
或调用MV_SequenceCaptureStop时调用。

此回调函数的原型为:

VOID (WINAPI *CONTINUEGARBMECHANISM) (PVOID pData, PMV_IMAGEINFO pImage,
ULONG ImageNumber, ULONG wholeLength, PVOID pUserData)
(详见后面的自定义函数类型)

返回值: 返回TRUE成功; 返回FALSE失败。

4. MV_SequenceCaptureStop

原型: MVAPI LONG WINAPI MV_SequenceCaptureStop (HANDLE
hDevice)

说明: 停止对设备帧存的连续捕获, 可在任何希望打断连续采集的
时刻发出。

入口参数: hDevice : 设备句柄。

返回值: 返回TRUE成功; 返回FALSE失败。

5. MV_GetSequenceFrameAddress

原型: MVAPI PVOID WINAPI MV_GetSequenceFrameAddress (HANDLE
hDevice, ULONG FrameNo, PMV_IMAGEINFO pPropertiy)

说明: 根据帧号FrameNo返回该帧的地址和帧属性。

入口参数: hDevice : 设备句柄。
FrameNo : 连续帧的帧号, 以0为起始。
pPropertiy : 数据缓冲区内每个图像的描述。

返回值: 返回帧号为FrameNo的帧地址。NULL代表错误, 一般是因为
FrameNo大于分配的帧存。

例: 连续采集多幅图像 (例如100幅), 可在采集进行时中断采集, 这种

采集是一种异步的采集。

```
VOID CMYCapture::StopCapture(...)
{
    //中断采集
    MV_SequenceCaptureStop( hDev );
}

VOID CMYCapture::StartCapture(...)
{
    // 对设备分配100幅图像的帧存，用于捕获处理前的原始数据
    MV_AllocSequenceFrameMemory( hDev, 0, 100, BEFORE_PROCESS );
    // 启动连续捕获
    MV_SequenceCaptureStart( hDev, ContinueGarbCallback, this );
}

VOID CMYCapture::ProcessCaptureData( PVOID pData, PIMAGEINFO
                                     pImage, ULONG ImageNumber )
{
    //保存图片
    MV_SaveFile( "Test", JPEG, pData, pImage, ImageNumber, FALSE,
                TRUE, 50 );
    MessageBox( 0, "连续采集结束", "采集了ImageNumber幅" 0 );
}

// 连续采集回调函数
VOID WINAPI ContinueGarbCallback ( PVOID pData, PIMAGEINFO pImage,
                                   ULONG ImageNumber, ULONG wholeLength, PVOID pUserData );
{
    CMYCapture* pCapture = (CMYCapture*) pUserData;
    pCapture->ProcessCaptureData( pData, pImage, ImageNumber );
}
```

1.4.4 捕获操作函数

1. MV_CaptureSingle

原型: MVAPI PVOID WINAPI MV_CaptureSingle (HANDLE hDevice,
 BOOL IsProcess, PVOID pInBuff, ULONG BuffLen,
 PMV_IMAGEINFO pInfo)

说明: 捕获一幅图像的数据。

入口参数: hDevice : 设备句柄。

IsProcess : 捕获的图像是否经过处理(捕获的图像是经过显示处理后的或者原始图像数据),
 FALSE为不经过处理的原始的图像数据。

pInBuff : 用户指定的缓冲。

BuffLen : 用户指定的缓冲长度。

pInfo : 调用返回后为图像信息描述的结构。

返回值: 返回存有图像数据的缓冲区指针, 当pInBuff为NULL时使用内部的缓冲区。

例1: 捕获的图像是采集的图像, 未经过处理的:

```
MV_IMAGEINFO Info;
MV_SetDeviceParameter(CardIndex, SET_GARBIMAGEINFO,
(DWORD)(&Info) ); // 得到当前的图像信息
PVOID pBuf1 = malloc( Info.Length );
PVOID pBuf2 = MV_CaptureSingle( CardIndex, FALSE, pBuf1,
Info.Length, &Info );
// 图像数据在pBuf2中。if( pBuf1 != pBuf2 )说明使用内部缓冲
//否则pBuf1和pBuf2一致。使用内部缓冲时不要释放该内部缓冲。
free(pBuf1);
```

例2: 捕获的图像是采集的图像, 但经过处理的:

```
MV_IMAGEINFO Info;
MV_SetDeviceParameter( CardIndex, SET_DISPIMAGEINFO,
(DWORD)(&Info) ); //得到当前的图像信息
PVOID pBuf1 = malloc( Info.Length );
PVOID pBuf2 = MV_CaptureSingle( CardIndex, TRUE, pBuf1,
Info.Length, &Info );// 图像数据在pBuf2中。
free(pBuf1);
```


例3: 用户也可使用内部的缓冲来捕获一幅图像的数据:

```
MV_IMAGEINFO  Info;
PVOID  pImage = MV_CaptureSingle( CardIndex, TRUE/FALSE,
                                NULL, 0, &Info ); // 图像数据在pImage中
// free(pBuf1); 不要这样干!!!
```

2. MV_SetCallBack

原型: MVAPI BOOL WINAPI MV_SetCallBack (HANDLE hDevice,
CALLBACKFUNC pCallBack, PVOID pUserData, CALLBACKTYPE
CallType)

说明: 设置设备的采集用户回调函数; 在设备运行时每一幅图像到达时将会调用该用户回调函数。

入口参数: hDevice : 设备句柄。
pCallBack : 设备采集的用户回调。

此回调函数的原型为:

```
typedef BOOL (WINAPI *CALLBACKFUNC)( PVOID pData, PMV_IMAGEINFO  
pImageInfo, PVOID pUserData, ULONG Index )
```

(详见自定义函数类型, 当用户回调返回FALSE时代表用户不希望系统显示图像, 返回TRUE代表用户希望显示。)

pUserData : 用户传递给回调函数的上下文数据。

CallType : 参见自定义枚举CALLBACKTYPE。

返回值: 返回TRUE成功; 返回FALSE失败。

注意: 1. 在用户设置的回调函数中, 如果用户的回调返回FALSE时将关闭显示但捕获仍在运行。这是去掉显示很好的办法。

2. 具体使用方法请参见演示程序的源代码。

1.4.5 存储图像文件函数

1. MV_SaveFile

原型: MVAPI BOOL WINAPI MV_SaveFile(PCHAR FileName,
MV_FILETYPE FileType, PVOID pImageData, PMV_IMAGEINFO
pImage, ULONG ImageTotal, BOOL IsUpDown, BOOL ColororNot,
ULONG Quality, BOOL m_bRGB15=TRUE)

说明: 保存pImageData中图像到MV_FILETYPE指定类型的文件。

入口参数: FileName : 文件名。

FileType : 文件类型RAW:原始数据文件; BMP:BMP文件;
JPEG:JPEG文件。

pImageData : 数据的缓冲指针。

ImageTotal : pImageData中包含图像的数目。

pImage : 每个图像的描述。

IsUpDown : 存文件时是否上下颠倒。

ColororNot : 存JPEG文件时是否要颜色。

Quality : 存JPEG文件时的质量0—99。

m_bRGB15 : 将16bit数据存为JPEG文件。由于目前采

用的JPEG压缩算法只支持8bit、24bit和32bit, 设置这个参数为TRUE时可以将16bit的rgb555彩色数据由软件转换为24bit的彩色数据然后存储为JPEG文件;

返回: 返回 TRUE 成功; 返回 FALSE 失败。

2. MV_SaveFilePro

原型: MVAPI BOOL WINAPI MV_SaveFilePro (PCHAR FileName,
MV_FILETYPE FileType, PVOID pImageData, PMV_IMAGEINFO
pImage, ULONG ImageTotal, BOOL IsUpDown, BOOL ColororNot,
ULONG Quality, BOOL m_bRGB15=TRUE)

说明: 将采集的单场图像数据通过软件缩放方式扩张为一帧数据, 将图像保存到MV_FILETYPE指定类型的文件。

入口参数: FileName : 文件名。

FileType : 文件类型RAW:原始数据文件; BMP:BMP文件;
JPEG:JPEG文件。

pImageData : 数据的缓冲指针。

| | | |
|------------|---|---------------------|
| ImageTotal | : | pImageData中包含图像的数目。 |
| pImage | : | 每个图像的描述。 |
| IsUpDown | : | 存文件时是否上下颠倒。 |
| ColororNot | : | 存JPEG文件时是否要颜色。 |
| Quality | : | 存JPEG文件时的质量0—99。 |
| m_bRGB15 | : | 将16bit数据存为JPEG文件。 |

返回：返回 TRUE 成功；返回 FALSE 失败。

1.4.6 16 位图像转 24 位图像函数

MV_RGB16TORGB24

原型: MVAPI BOOL WINAPI MV_RGB16TORGB24 (BYTE* pRGB16Buf,
long lWidth,
long lHeight,
BYTE* pRGB24Buf,
BOOL RGB1516);

说明: 将16bit的图像转换为24bit的图像。

参数说明: pRGB16Buf: 16bit 图像数据指针;

lWidth: 输入图像宽度;

lHeight: 输入图像高度;

pRGB24Buf : 24bit 图像数据指针;

RGB1516: TRUE表示输入图像数据为RGB15位,

FALSE表示输入图像数据为RGB565。

返回: 返回 TRUE 成功; 返回 FALSE 失败。

1.4.7 场扩帧函数

MV_AmplifyImage

原型: MVAPI BOOL WINAPI MV_AmplifyImage (BYTE* pInputImage,
long lWidth,
long lHeight,
BYTE* pOutputImage,
int Bits);

说明: 将获得的一场图像转换为一帧大小的图像。

入口参数: pInputImage: 输入图像数据指针;
lWidth: 图像宽度;
lHeight: 图像高度;
pOutputImage: 输出图像数据指针;
Bits: 比特数, 8, 16, 24 或 32 位。

返回: 返回 TRUE 成功; 返回 FALSE 失败。

1.4.8 错误提示函数

MV_GetLastError

原型: MVAPI DWORD WINAPI MV_GetLastError(BOOL
bDisplayErrorStyring)

说明: 错误提示。每一次函数调用后如果调用返回不正确你可以调用该函数来得到错误原因。

入口参数: bDisplayErrorStyring : 设为TRUE时系统将为用户弹出一个对话框来显示错误提示串; 设为FALSE时系统将返回错误代码。

返回值:

- 0: 没有任何错误, 函数调用成功可以进行后续操作;
- 1: 该卡已经被创建过;
- 2: 该卡所需的库不存在;
- 3: 该下标的设备不能被创建;
- 4: 无效的设备句柄;
- 5: 该下标的设备未被创建;
- 6: 没有足够的资源;
- 7: 没有分配大内存;
- 8: 该卡所需的库已损坏。

1.4.9 OSD 操作函数

OSD是一种硬件操作，OSD的位屏蔽图中的每一位代表一个像素。

1. MV_SetMaskFunction

原型: MVAPI BOOL WINAPI MV_SetMaskFunction(HANDLE hDevice, ULONG OsdMode)

说明: 按指定的模式操作板卡的硬件OSD(掩码)功能。一般在OSD图案屏蔽位发送到设备之后，用于设置OSD模式。

入口参数: hDevice : 设备句柄;
OsdMode : 设置OSD的模式。值为0: 无OSD; 值为1: 0屏蔽OSD; 值为2: 1屏蔽OSD。

返回值: 返回TRUE成功; 返回FALSE失败。

2. MV_MakeMaskBit

原型: MVAPI LPBYTE WINAPI MV_MakeMaskBit(HANDLE hDevice, HWND hWnd, RECT Area, COLORREF Color)

说明: 用于生成一个OSD屏蔽位的图案。用户可将一幅图像先在一个窗口中显示出来，然后将该窗口句柄和希望在这个窗口中截取的位置区域传入该函数，函数返回图案屏蔽位的缓冲指针。

入口参数: hDevice: 设备句柄。
hWnd : 图像所在的窗口句柄。
Area : 窗口中希望截取的位置和区域。
Color : 由于硬件OSD特性，该Color用于将屏蔽位图案进行二值化处理。生成用于屏蔽/不屏蔽你感兴趣的顏色。

返回值: 返回OSD图案屏蔽位的缓冲指针。

注意: 用户使用该函数时，若将hDevice参数设为NULL，将返回OSD图案区域(Area指出)所占的字节数。可以参见后面例子。

3. MV_SetMaskBit

原型: MVAPI BOOL WINAPI MV_SetMaskBit(HANDLE hDevice, RECT MaskArea, LPBYTE pBitPattern)

说明: 将OSD图案屏蔽位发送到设备。

入口参数: hDevice : 设备句柄。

MaskArea : OSD图案屏蔽位作用在硬件捕获的图像上的位置和区域。

pBitPattern: OSD图案屏蔽位的缓冲指针。

返回值: 返回TRUE成功; 返回FALSE失败。

例:将hWnd所指的画布上以(64, 64)为起始点, (300, 300)为结束点的图案作模版, 叠加到当前图像以(128, 128)为起始点的区域。

```
RECT    Area = { 64, 64, 300, 300 };
LPBYTE  pOSD = MV_MakeMaskBit ( hDev, hWnd, Area ,
                                COLORREF(0, 200, 0)); // 得到OSD的位屏蔽图
Area = {128, 128, (128+300-64), (128+300-64)};
MV_SetMaskBit( hDev, Area, pOSD );
MV_SetMaskFunction( hDev, 2 );
```

例:用户可连续生成几个图案屏蔽位, 各有不同的图案, 然后将它们各自轮流发送到设备, 用户不用担心会与采集不同步。

```
LPBYTE  g_pOSD[10];          // 10个全局图案
        .....
RECT    Area  = { 64, 64, 300, 300 };
ULONG   Length = (ULONG) MV_MakeMaskBit (NULL, NULL, Area,
                                           COLORREF(0, 200, 0));
// 生成10个图案屏蔽位
for( int i=0; i<10; i++ )
{
    if( g_pOSD[i] ) free( g_pOSD[i] );
    g_pOSD[i] = NULL;
    g_pOSD[i] = malloc( Length );
    // MY_LoadPictureToWindowdown函数是您自己写的, 用于将图
    像贴到hWnd所指的窗口中。
    MY_LoadPictureToWindowdown( hWnd, ... );
    LPBYTE  pTemp = MV_OSDCyclostyle( m_hDev, hWnd, Area );
    memcpy( g_pOSD[i], pTemp, Length );
}
MV_SetMaskFunction( m_hDev, 2 );
```



```
.....  
RECT  OsdArea = { 128, 128, (128+300-64), (128+300-64)};  
for( int i=0; i<10; i++ )  
{  
    if( g_pOSD[i] == NULL ) continue;  
    Sleep( 1000 );  
    MV_SetMaskBit( m_hDev, OsdArea, g_pOSD[i] );  
}  
.....
```

1.4.10 外触发操作函数

下面介绍输入输出触发控制函数。各种类型的板卡有不同的输入输出管脚定义，具体请参看硬件说明。

1. MV_SetOutputState

原型: MVAPI BOOL WINAPI MV_SetOutputState(HANDLE hDevice,
ULONG Index, ULONG HorL)

说明: 使板卡输出一个高电平或者低电平;

入口参数: hDevice : 设备句柄。

Index : 指明第几路输出。

HorL : 指明输出低电平0, 还是高电平1。

返回值: 返回TRUE成功; 返回FALSE失败。

2. MV_GetInputState

原型: MVAPI ULONG WINAPI MV_GetInputState(HANDLE hDevice,
ULONG Index);

说明: 用户主动的读取输入状态;

入口参数: hDevice : 设备句柄。

Index : 指明第几路输入。

返回值: 返回0代表低电平; 返回1代表高电平。

3. MV_SetInputCallBack

原型: MVAPI BOOL WINAPI MV_SetInputCallBack (HANDLE hDevice,
ULONG Index, ULONG UniquelyID, PTRIGGERROUTINE
pTirggerCall, PVOID pContext);

说明: 为输入触发管脚设置用户回调函数。一旦有外输入触发产生, 回调函数将被调用, 当pTirggerCall为NULL时将不能触发。

入口参数: hDevice : 设备句柄。

Index : 指明管脚为0或1。

UniquelyID: 用户为该输入外触发设置的唯一标示符;
当有外输入触发产生, 回调函数被调用时,
它将传递给用户回调函数。

pContext : 为用户传递给回调函数的上下文数据。

pTirggerCall: 为输入触发回调函数。

此回调函数的原型为：

**VOID (WINAPI *PTRIGGERROUTINE) (ULONG UniquelyID, ULONG Reson,
PVOID pContext)**

(详见自定义函数类型)

返回值： 返回TRUE成功；返回FALSE失败，原因很可能是Index超过范围。

详细使用方法可参考演示程序的源代码。

1.4.11 获得图像数据

1. MV_ReadPixel

原型: MVAPI ULONG WINAPI MV_ReadPixel (PMV_IMAGEINFO
pProperty, PVOID pImageData, PPOINT Point, PBYTE pVal)

说明: 从图像缓冲区中读一个像素的值。

入口参数: pProperty : 数据缓冲区内图像的描述。

pImageData : 图像的数据缓冲区。

Point : 像素在图像中的位置。

pVal : 调用返回后存放像素的值。

返回值 : 返回像素所占的字节数。

2. MV_WritePixel

原型: MVAPI VOID WINAPI MV_WritePixel (PMV_IMAGEINFO
pProperty, PVOID pImageData, PPOINT Point, PBYTE pVal)

说明: 写一个像素的值到图像缓冲区中。

入口参数: pProperty : 数据缓冲区内图像的描述。

pImageData : 图像的数据缓冲区。

Point : 像素在图像中的位置。

pVal : 写入的像素的值。

返回值 : 无。

3. MV_ReadLine

原型: MVAPI ULONG WINAPI MV_ReadLine (PMV_IMAGEINFO
pProperty, PVOID pImageData, ULONG nLine, PBYTE pVal)

说明: 从图像缓冲区中读一行的数据。

入口参数: pProperty : 数据缓冲区内图像的描述。

pImageData : 图像的数据缓冲区。

nLine : 行在图像中的行号, 以零为下标。

pVal : 调用返回后存放行的数据。

返回值 : 返回行所占的字节数。

4. MV_WriteLine

原型: MVAPI VOID WINAPI MV_WriteLine(PMV_IMAGEINFO pProperty,
PVOID pImageData, ULONG nLine, PBYTE pVal)

说明: 写一行的数据到图像缓冲区。

入口参数: pPropertiy : 数据缓冲区内图像的描述。
pImageData : 图像的数据缓冲区。
nLine : 行在图像中的行号, 以零为下标。
pVal : 写入行的数据。

返回值 : 无。

5. MV_ReadArea

原型: MVAPI ULONG WINAPI MV_ReadArea(PMV_IMAGEINFO pProperty,
PVOID pImageData, PRECT pRect, PBYTE pVal)

说明: 从图像缓冲区中读一块区域的数据。

入口参数: pPropertiy : 数据缓冲区内图像的描述。
pImageData : 图像的数据缓冲区。
pRect : 块区域在图像中的RECT结构的指针。
pVal : 调用返回后存放块区域的数据。

返回值 : 返回该块区域所占的字节数。

6. MV_WriteArea

原型: MVAPI VOID WINAPI MV_WriteArea(PMV_IMAGEINFO
pProperty, PVOID pImageData, PRECT pRect, PBYTE pVal)

说明: 写一块区域的数据到静态图像。

入口参数: pPropertiy : 数据缓冲区内图像的描述。
pImageData : 图像的数据缓冲区。
pRect : 块区域在图像中的RECT结构的指针。
pVal : 写入块区域的数据。

返回值 : 无。

1. 4. 12 自动帧测行场频

Levin M20/RGB20/VGA卡支持自动帧测行场频功能。

1. MV_TestSignal

原型: MVAPI BOOL WINAPI MV_TestSignal (HANDLE hDevice, DWORD XSize, DWORD YSize)

功能: 开始自动帧测输入信号。

入口参数: hDevice : 设备句柄。

Xsize : 输入的有效的行内点数。

Ysize : 输入的有效的场内行数。

返回值: 返回TRUE成功; 返回FALSE失败。

2. MV_GetSignalParam

原型: MVAPI DWORD WINAPI MV_GetSignalParam (HANDLE hDevice, VIDEOSIGNAL Signal, float *FloatVal, DWORD *IntVal)

功能: 得到自动帧测信号参数的值。

入口参数: hDevice : 设备句柄。

Signal : 信号参数的类型。

IntVal : 调用返回后存放整型信号参数的值。

FloatVal : 调用返回后存放浮点型信号参数的值。

返回值: 0: 失败;

1: 代表IntVal内数据有效;

2: 代表FloatVal内数据有效。

3. MV_SaveSignalParamToIni

原型: MVAPI BOOL WINAPI MV_SaveSignalParamToIni (HANDLE hDevice)

功能: 将自动帧测的参数保留。

入口参数: hDevice : 设备句柄。

返回值: 返回TRUE成功; 返回FALSE失败。

1. 4. 13 RGB 分量显示

原型: MVAPI BOOL WINAPI MV_SplitRGB (HANDLE hDevice, PMV_RGB pRGB, ULONG DispType, ULONG DispPolicy, ULONG CallBackPolicy)

功能: 将图像的RGB分离, 然后进行采集或显示。

入口参数: hDevice : 设备句柄。

PRGB: 为指向MV_RGB结构的指针。该结构指明R, G, B每个量在显示时的窗口和窗口中的位置, 用户在调用前必须填充该结构。

DispType: 为显示的类型, 由用户选一个最适合本机的类型, 可选值为0, 1。

DispPolicy: 为显示的策略, 即用户可指定显示B_CHANNEL或G_CHANNEL或R_CHANNEL中的一个分量或全部或部份(即B_CHANNEL, G_CHANNEL, R_CHANNEL的位或)。

分量定义如下:

```
#define B_CHANNEL    0x00000001
#define G_CHANNEL    0x00000002
#define R_CHANNEL    0x00000004
```

CallBackPolicy: 为设置回调的策略, 即用户可指定选择分量或全部或部份。

返回值: 返回TRUE成功; 返回FALSE失败。

注意: 当CallBackPolicy指定为BEFORE_PROCESS时, 回调返回的数据仍然为图像原始的的RGB数据。当CallBackPolicy指定为AFTER_PROCESS时, 回调返回的数据为分量分离后的数据, 具体是哪一个分量由其回调函数的最后一个参数Index指明。指明如下: 0x80000000时pData代表B_CHANNEL数据, 0x80000001时pData代表G_CHANNEL数据, 0x80000002时pData代表R_CHANNEL数据。

例: 用户希望将R, G, B数据显示在三个独立的窗口中, 并且提取每一路分量的数据。

```
VOID CMYClass::UsedRGBSplit()
{
    // 为处理前/后都设置一个用户回调
    MV_SetCallBack( hDev, BEFORE_CallBackFunction, this, BEFORE_PROCESS );
    MV_SetCallBack( hDev, AFTER_CallBackFunction, this, AFTER_PROCESS );
}
```

```
MV_RGB rgb; //声明MV_RGB结构
// 分别对应三个窗口的显示区域
rgb.R_hWnd = hWnd_R;      rgb.R_Rect = { 0, 0, 352, 288 };
rgb.G_hWnd = hWnd_G;      rgb.G_Rect = { 10, 10, 362, 298 };
rgb.B_hWnd = hWnd_B;      rgb.B_Rect = { 20, 20, 372, 308 };
MV_SplitRGB ( hDev, &rgb, 0, R_CHANNEL|G_CHANNEL|B_CHANNEL,
AFTER_PROCESS );

return;
}

BOOL WINAPI CMYClass::AFTER_CallBackFunction ( PVOID pData, PMV_IMAGEINFO
pImageInfo, PVOID pUserData, ULONG Index );
{
    CMYClass* pthis = (CMYClass*)pUserData;
    if ( Index == R_CHANNEL )
    { // 当前的数据是红路，对红路数据进行处理。 }
    if( Index == G_CHANNEL )
    { // 当前的数据是绿路。对绿路数据进行处理。 }
    if( Index == B_CHANNEL )
    { // 当前的数据是蓝路。对蓝路数据进行处理。 }
    Return TRUE;
}

BOOL WINAPI CMYClass::BEFORE_CallBackFunction ( PVOID pData, PMV_IMAGEINFO
pImageInfo, PVOID pUserData, ULONG Index )
{
    CMYClass* pthis = (CMYClass*) pUserData;
    //pData中数据仍然为图像原始的未分离的RGB数据;
    Return TRUE;
}
```


1. 4. 14 自定义函数类型

1. 原型: typedef BOOL (WINAPI *CALLBACKFUNC)(PVOID pData, PMV_IMAGEINFO pImageInfo, PVOID pUserData, ULONG Index)

说明: 设备的采集用户回调函数原型，当设置回调后，只要设备处于运行状态，每到一场/帧数据到达时该用户回调被调用。该种回调分两种回调，在设置回调函数中加以区别。第一种为原始数据的回调，pData返回的是原始的未经处理的被捕获得图像数据。第二种为经处理后的回调，pData返回的是经处理捕获得图像数据，由于经过处理，它不能实时的在每一帧/场到达时被调用，因而只适用对实时无要求的场合，但是它可在回调中做很多的事情而不必担心拉慢系统，这个特性是第一种回调没有的。第一种回调是一种实时的回调，因此不能在回调中做太多的事情，一般用户可用的时间上限是对于按场采集小于一场的时间，对于按帧采集小于一帧的时间。

入口参数: pData : 为当前图像的数据指针。
pImageInfo: 当前图像描述结构的指针，描述pData中数据图像的属性。
pUserData : 为用户传递给回调函数的上下文数据。
Index : 当设备设为按场采集时：为1代表奇场，0代表偶场。当设备设为按帧采集时无意义。

返回值: 当用户回调返回FALSE时代表用户不希望显示图像，TRUE代表用户希望显示图像。

注意:如果你的回调返回FALSE时将关闭显示，但捕获仍在运行这是去掉显示很好的办法。对于该类型的回调函数中你不该处理过多的事情，否则会影响系统的采集工作引发副作用。

(参见: 函数MV_SetCallBack和枚举CALLBACKTYPE)

例: 第一种采集回调:

```
BOOL WINAPI CallBack( PVOID pData, PMV_IMAGEINFO pImageInfo,
                     PVOID pUserData, ULONG Index );
{
    // 用户的工作，占用的时间必须小于一帧/场的时间。
    return TRUE; // FALSE;
}
```

```

.....
MV_SetCallBack( m_hDevice, CallBack, pUserData, BEFORE_PROCESS );
.....

第二种采集回调：
BOOL WINAPI CallBack ( PVOID pData, PMV_IMAGEINFO pImageInfo,
                      PVOID pUserData, ULONG Index );
{
    // 用户的工作，占用的时间无限制
    return TRUE; // FALSE;
}

.....
MV_SetCallBack( m_hDevice, CallBack, pUserData, AFTER_PROCESS );
.....

```

2. 原型： typedef VOID (WINAPI *PTRIGGERROUTINE) (ULONG UniquelyID, ULONG Reson, PVOID pContext)

功能： 输入触发用户回调函数原型，一旦当外输入触发产生被设置后，该回调函数将会被调用，以通知用户由外触发信号的到达。外触发信号的类型由Reson来区别。

入口参数：

UniquelyID: 用户设置的唯一标示符，用以标示是多个输入触发中的哪一个被触发。

Reson : 当前触发是低电平0，还是高电平1。

pContext : 为用户传递给回调函数的上下文数据。

返回值： 无。

(参见: MV_SetInputCallBack。)

例： VOID WINAPI TriggerCallBack(ULONG UniquelyID, ULONG Reson, PVOID pContext)

```

{
    if( My_ID != UniquelyID ) return;
    if( Reson == 0 )
    {
        // 下降沿触发，当前是低电平
    }
    Else

```

```
    {  
        // 上升沿触发，当前是高电平  
    }  
    return;  
}  
  
.....  
MV_SetInputCallBack( m_hDevice, Index, My_ID, TriggerCallBack, NULL );  
.....
```

3. 原型: typedef VOID (WINAPI *CONTINUEGARBMECHANISM) (PVOID pData, PMV_IMAGEINFO pImage, ULONG ImageNumber, ULONG wholeLength, PVOID pUserData)

说明: 连续采集机制用户回调函数原型。当开始进行连续捕获的条件满足，即在分配时指定的图像数目已经到达时，或用户发出了停止命令，该用户回调将会被调用。

入口参数: pData : 数据缓冲区。

pImage : 数据缓冲区内图像的描述。

ImageNumber : 连续捕捉了多少幅图，一般为用户程序设置的帧存数目；当用户程序连续捕获运行时在发出了停止命令情况下为真实已经采集的图像数目。

wholeLength : 数据缓冲区的全长以字节记。

pUserData : 用户传递给回调函数的数据。

返回值: 无。

(参见: MV_StartSequenceCapture)

4. 原型: typedef VOID (WINAPI *GDIOPERATIONFUNC) (PGDIOPERATION pOper, PVOID pVal1, PVOID pVal2, PVOID pVal3)

说明: 使用图形操作在采集图像上叠加字符串, 线段和圆, 利用这个功能可生成不轨则图形并将它迭加到图像。

入口参数: pOper : 图形体操作结构。

pVal1, pVal2, pVal3 : 不同操作体函数, 其设置不同。

返回值: 无。

(参见自定义结构类型: PGDIOPERATION)

注意: 图形体的操作是建立在软件完成的基础上的, 因而在使用图形体前你必须将缓冲模式设置为SYSTEM_MEMORY_GDI方式, 并且得到图形体操作结构来操作图形体。如果你想采集的图像上也有这些图形体, 那么在采集时必须使用AFTER_PROCESS方式。

下面的两行代码是必须的;

```
// 判断当前的缓冲模式, 如果需要则设置
if ( MV_GetDeviceParameter(hDevice, BUFFERTYPE) != SYSTEM_MEMORY_GDI )
{
    MV_SetDeviceParameter( hDevice, BUFFERTYPE, SYSTEM_MEMORY_GDI );
}
// 得到图形体操作结构
PGDIOPERATION pOper =
(PGDIOPERATION)MV_GetDeviceParameter(hDevice, GET_GRAPHICAL_INTERFACE );
// pOper你应该保留决不能改变, 它将在每一次调用图形体操作函数时被使用。
```

该GDIOPERATION中的成员大多为指向函数的指针。有设置输出字符串及其颜色, 字体格式, 显示位置; 设置图形及其画笔; 设置按类擦除图形以及擦除所有图形。选择不同的操作, pVal1, pVal2, pVal3的设置不同, 具体如下:

a. SetGDIText

设置输出的字符串, 将字符串叠加到图像上。pVal1为以null结束的字符串, pVal2为第几个字符串的下标的指针, pVal3为NULL。

```
char str[] = "Hello World";
int Index = 2; //可叠加多个字符串, 每个有其下标 (0代表叠加时间, 1
               代表叠加日期), 这里设为第三个
```

```
pOper->SetGDIText( pOper, (PVOID)str, (PVOID)(&Index), NULL );
```

b. SetGDITextColor

设置输出的字符串的颜色。pVal1为颜色值指针，pVal2为第几个字符串的下标的指针，pVal3为NULL。

```
COLORREF crColor = RGB( 255, 0, 0 );    // 设置字符串的颜色为红色。
int Index = 2;
pOper-> SetGDITextColor ( pOper, (PVOID)(&(PVOID) crColor),
                        (PVOID)(&Index), NULL );
```

c. SetGDITextFormat

设置输出的字符串的字体格式，当调用该函数时可以选择是否弹出一个标准的设置字体格式的选择对话框，如果选择弹出，用户可以通过该对话框选择字体和格式。

pVal1为第几个字符串的下标的指针，pVal2为设置字体的大小，pVal3为是否弹出设置字体格式的对话框，1为弹出，Null为不弹出。

```
int Index = 2;
int Size = 12;
pOper-> SetGDITextFormat( pOper, (PVOID)(&Index), (PVOID)(&Size),
                        NULL );
```

d. SetGDITextPosition

设置输出的字符串相对于图像的位置。pVal1和pVal2的X坐标和Y坐标，pVal3为第几个字符串的下标的指针。

例：将第10个字符串挪到（352，288）开始的地方。

```
int X_off = 352;
int Y_off = 288;
int Index = 10;
pOper->SetGDITextPosition( pOper, (PVOID)(&X_off), (PVOID)(&Y_off),
                        (PVOID)(&Index));
```

e. SetGDIGraph

设置输出图形的类型以及左上点和右下点。pVal1的内容若为0代表直线，1代表圆； pVal2为POINT数组类型的指针，该数组有两个POINT类型组成。pVal3为第几个直线或圆的下标的指针。

例：在图像的（12，34）到（567，765）之间画一条线和一个圆。

```

POINT Start = { 12, 34 };
POINT End  = { 567, 765 };
POINT Po[2] = { Start, End };
int Index = 0;
int type = 0;      // 画一条线
pOper->SetGDIGraph( pOper, (PVOID)(&type),
(PVOID)(&Po), (PVOID)(&Index));
Index = 1;
type = 1;          // 画一个圆
pOper->SetGDIGraph( pOper, (PVOID)(&type), (PVOID)(&Po),
(PVOID)(&Index));

```

f. SetGDIGraphPen

设置输出图形的画笔。pVal1为画笔样式的指针，pVal2为画笔宽度的指针，pVal3为画笔颜色的指针。该函数中画笔样式，宽度，颜色参考MSDN的CreatePen API函数。

例：设置画笔的样式为点画线，宽度为2个像素，颜色为红色。

```

int      Style = PS_DOT;
int      Width = 2;
COLORREF Color = RGB( 255, 0, 0 );
pOper->SetGDIGraphPen( pOper, (PVOID)(&Style), (PVOID)(&Width),
(PVOID)(&Color) );

```

g. SetGDICanCalAll

擦除所有先前所画的直线或圆。pVal1的内容若为0代表字符串，1代表直线，2代表圆；pVal2，pVal3为NULL。

例：分别擦除所有的字符串，所有的直线，所有的圆。

```

int type = 0;  // 擦除所有的字符串
pOper->SetGDICanCalAll( pOper, (PVOID)(&type), NULL, NULL );
type = 1;     // 擦除所有的直线
pOper->SetGDICanCalAll( pOper, (PVOID)(&type), NULL, NULL );
type = 2;     // 擦除所有的圆
pOper->SetGDICanCalAll( pOper, (PVOID)(&type), NULL, NULL );

```

h. SetGDICanCalOne

擦除指定类型及下标的图形体。 pVal1的内容若为0代表字符串, 1代表直线, 2代表圆; pVal2为指定图形体的下标; pVal3为NULL。

例：分别擦除第2个字符串, 第3个圆, 第5个直线。

```
int type = 0;
int Index = 5; // 擦除第5个字符串
pOper->SetGDICanCalOne( pOper, (PVOID)(&type), (PVOID)(&Index),
NULL );
type = 1; Index = 5; // 擦除第5条直线
pOper->SetGDICanCalOne( pOper, (PVOID)(&type), (PVOID)(&Index),
NULL );
type = 2; Index = 5; // 擦除第5个圆
pOper->SetGDICanCalOne( pOper, (PVOID)(&type), (PVOID)(&Index),
NULL );
```

实现不同颜色的图形叠加，步骤如下：

- (1) 在叠加直线的程序中，可以叠加 100 个（0—99）不同颜色的直线，但只有低 16 部分时支持颜色部分的，示例代码如下：

```
PGDIOPERATION g_pGDI; // 叠加第一条直线
int nGraphType=0;
int Style=PS_DOT; //可根据 CPen 参数 nPenStyle 的选项进行更改设置
int Width=2;
COLORREF Color1 = RGB( 255, 0, 0 );
POINT xLine[2], start, end;
start.x = 100; start.y = 100;
end.x = 500; end.y = 600;
xLine[0] = start; xLine[1]= end;
ULONG nLineCount = 0; // nLineCount 代表叠加直线的个数, 0 代表第一条
g_pGDI1=(PGDIOPERATION)MV_GetDeviceParameter(CurDevice,
GET_GRAPHICAL_INTERFACE); // 得到直线的操作结构
g_pGDI1->SetGDIGraph(g_pGDI1,&nGraphType,&xLine,&nLineCount); //叠
加直线到图像上
g_pGDI1->SetGDIGraphPen( g_pGDI1, (PVOID)(&Style), (PVOID)(&Width),
(PVOID)(&Color1) ); //设置直线的样式, 宽度, 颜色
```

```

nLineCount++; //这里代表叠加第二条直线
int nGraphType=0;
int Style=PS_DOT|(1<<16); //代表第二条直线
int Width=2;
COLORREF Color1 = RGB( 0, 255, 0 );
其它代码如上
int nGraphType=0;
int Style=PS_DOT|(99<<16); //代表第 100 条直线
int Width=2;
COLORREF Color1 = RGB( 0, 0, 255);
其它代码如上

```

- (2) 在叠加椭圆的程序中，可以叠加 100 个（0—99）不同颜色的椭圆形，但只有高 16 位部分是支持颜色的部分，示例代码如下：

```

int nGraphType=1; // 叠加第一个椭圆
int Style=PS_DOT|((100)<<16); //代表第一个椭圆
int Width=2;
COLORREF Color1 = RGB( 0, 255, 0 );
PGDIOPERATION g_pGDI2;
POINT xLine[2], start, end;
start.x=150; start.y=150;
end.x=400; end.y=450;
xLine[0]=start; xLine[1]=end;
ULONG nCirCount = 0; //代表叠加圆的个数
g_pGDI2=(PGDIOPERATION)MV_GetDeviceParameter(CurDevice, GET_GRAPHICA
L_INTERFACE); //得到圆的操作结构
//叠加圆到图像上
g_pGDI2->SetGDIGraph(g_pGDI2, &nGraphType, &xLine, &nLineCount);
g_pGDI2->SetGDIGraphPen(g_pGDI2, (PVOID)(&Style), (PVOID)(&Width),
(PVOID)(&Color)); //设置圆的的样式，宽度，颜色
nCirCount++; //这里代表叠加下一个圆

```


第二部分 配置文件说明

2.1 MV_MX0.INI

对应采集卡：Levin M10、M20、RGB10、RGB20

配置文件名：MV_MX0.INI

配置文件说明：

[POLARITY]：该段定义了信号同步极性的策略。

IN_ROW：输入信号的行同步的极性。0：正极性；1：负极性，无对应SDK的MV_PARAMTER枚举参数。

IN_FIELD：输入信号的场或帧同步的极性。0：正极性；1：负极性，无对应SDK的MV_PARAMTER枚举参数。

OUT_ROW：输出信号的行同步的极性。0：正极性；1：负极性，无对应SDK的MV_PARAMTER枚举参数。

OUT_FIELD：输出信号的场或帧同步的极性。0：正极性；1：负极性，无对应SDK的MV_PARAMTER枚举参数。

OUT_COMPLX：输出复合信号的同步的极性。0：正极性；1：负极性，无对应SDK的MV_PARAMTER枚举参数。

[DISPLAY]：该段定义了图像的显示参数。

RESET_TIME：定义一帧或一场之间的最大的时间间隔。它应该比帧和帧或场和场之间的时间间隔大一些，无对应的MV_PARAMTER枚举参数。

DISPLAY_Left：图像的显示在窗口的X方向的起始偏移，对应SDK的MV_PARAMTER枚举参数DISP_LEFT。

DISPLAY_Top：图像的显示在窗口的Y方向的起始偏移，对应SDK的MV_PARAMTER枚举参数DISP_TOP。

DISPLAY_Height：图像显示时的显示高度，对应SDK的MV_PARAMTER

枚举参数DISP_HEIGHT。

DISPLAY_Width: 图像显示时的显示宽度, 对应SDK的MV_PARAMTER枚举参数DISP_WIDTH。

DISPLAY_IsFilp: 图像在显示时是否左右翻转, 对应SDK的MV_PARAMTER枚举参数DISP_WIDTH。

DISPLAY_Mode: 选用的显示方式, 同时也是采集时选用的缓冲类型, 对应SDK的MV_PARAMTER枚举参数BUFFERTYPE。

DISPLAY_FilpMODE: 当信号为隔行信号时当左右翻转时两场的翻转趋向, 0两场翻转, 1偶翻奇不翻, 2奇翻偶不翻无对应SDK的MV_PARAMTER枚举参数。

[MV_M10] : 该段对应M10卡的捕获时的硬件相关的设置;

[MV_M20] : 该段对应M20卡的捕获时的硬件相关的设置;

[MV_RGB10] : 该段对应RGB10卡的捕获时的硬件相关的设置

[MV_RGB20] : 该段对应RGB20卡的捕获时的硬件相关的设置。

IMAGE_bField: 隔行信号时有效, 捕获到每一场的数据, 时间间隔为帧的时间间隔的一半, 对应SDK的MV_PARAMTER枚举参数WORK_FIELD。

IMAGE_LP, IMAGE_VCOG, IMAGE_VCOR, IMAGE_GLLevel: 和信号相关的卡的硬件设置。对于M10, RGB10出厂时固定不变, 无需用户调节; 对于M20, RGB20对应自动侦测的VIDEOSIGNAL枚举参数GLLEVEL, LP, VCOG, VCOR参数。自动侦测之后用户可选择保留, 将修改这些设置。

IMAGE_Bit: 设置采集图像的位深度, 对应SDK的MV_PARAMTER枚举参数GARB_BITDESCRIBE。

IMAGE_UpEnd: 图像采集时是否用硬件上下翻转, 对应SDK的MV_PARAMTER枚举参数WORK_UPDOWN。

IMAGE_IsSkip: 对于隔行信号, 图像采集时是否用硬件进行奇偶场行的穿插, 对应SDK的MV_PARAMTER枚举参数WORK_SKIP。

IMAGE_Synchro: 同步信号的选择, 对应SDK的MV_PARAMTER枚举参数WORK_SYNC。

IMAGE_XOffset, IMAGE_YOffset: 设置视频信号的起始位置, 对应SDK的MV_PARAMTER枚举参数GARB_XOFF, GARB_YOFF。

IMAGE_YSize, IMAGE_XSize: 设置视频信号的大小尺寸, 对应SDK的MV_PARAMTER枚举参数GARB_HEIGHT, GARB_WIDTH。

IMAGE_Channel: 设置视频信号的来源的通道, 对应SDK的MV_PARAMTER枚举参数ADJUST_CHANNEL。

IMAGE_Interlace: 视频信号是否是隔行或逐行信号, 对应SDK的MV_PARAMTER枚举参数WORK_INTERLACE, 和自动侦测的VIDEOSIGNAL枚举参数ISINTERLACE。

IMAGE_LineTotal: 设置信号的行总数, 对应 SDK 的 MV_PARAMTER 枚举参数 GARB_WHOLEWIDTH, 和自动侦测的 VIDEOSIGNAL 枚举参数 DIVIDER。

2.2 MV7146. INI

对应采集卡：V3A、V300、V500、V510、V520、Moka-C50、Moka-C51

配置文件名：MV7146. INI

配置文件说明：

[TIME_PROCESS]：该段定义了初始化采集卡时的判断时间

INITTIME：默认为200，如果出现初始化采集卡时提示无法判断采集卡类型，请将增加该值，可增加到500。

[DISPLAY]：该段定义了图像的显示/处理参数。

DISP_TOP：显示在窗口的Y方向的起始偏移，对应SDK的MV_PARAMTER枚举参数DISP_TOP。

DISP_LEFT：图像的显示在窗口的x方向的起始偏移，对应SDK的MV_PARAMTER枚举参数DISP_LEFT。

DISP_HEIGHT：图像显示时的显示高度，对应的MV_PARAMTER枚举参数为DISP_HEIGHT。

DISP_WIDYH：图像显示时的显示宽度，对应的MV_PARAMTER枚举参数为DISP_WIDTH。

FIELDTOFRAME：将场按帧进行扩展。无对应的MV_PARAMTER枚举参数。

FIELDTOFRAMEINDEX：按场采集时采集的场，1：奇场，0：偶场。无对应的MV_PARAMTER枚举参数。

STRINGNUM：图形体叠加时字符串的个数，无对应的MV_PARAMTER枚举参数。

LINENUM：图形体叠加时线的个数，无对应的MV_PARAMTER枚举参数。

ARCNUM：图形体叠加时圆的个数，无对应的MV_PARAMTER枚举参数。

[GARB_XXX]：捕获时的硬件相关的设置。为了区别不同类型的采集卡，用XXX对应不同的类型的采集卡，即不同类型的采集卡的设置各自占一个

段。目前XXX共有V3A, V300, V500, V510, V520五种。

GARB_BUFFERTYPE: 选用的显示方式, 同时也是采集时选用的缓冲类型, 对应SDK的MV_PARAMTER枚举参数BUFFERTYPE。

GARB_XOFF, GARB_YOFF: 设置视频信号的X起始位置, Y起始位置。对应SDK的MV_PARAMTER枚举参数GARB_XOFF, GARB_YOFF。

GARB_INPUT_HEIGHT, GARB_INPUT_WIDTH: 设置视频信号输入的高度, 宽度。对应MV_PARAMTER的参数GARB_IN_HEIGHT, GARB_IN_WIDTH。

GARB_HEIGHT, GARB_WIDTH: 设置视频信号输出的高度, 宽度 (即实际采集大小)。对应MV_PARAMTER的参数GARB_HEIGHT, GARB_WIDTH。

GARB_BITDESCRIBE: 图像采集时的采集的图像的位深度, 对应SDK的MV_PARAMTER枚举参数GARB_BITDESCRIBE。

GARB_OSDMODE : 图像采集时硬件的OSD的模式, 对应SDK的MV_PARAMTER枚举参数OSD_MODE。

GARB_ISFILP: 图像采集时硬件的左右翻转, 对应SDK的MV_PARAMTER枚举参数WORK_FLIP。

GARB_ISFIELD: 确定图像采集时是按场采集, 或按帧采集, 对应SDK的MV_PARAMTER枚举参数WORK_FIELD。

GARB_ISUPDOWN: 图像采集时是否用硬件上下翻转, 对应SDK的MV_PARAMTER枚举参数WORK_UPDOWN。

GARB_SWAP: 图像采集时的字节的endain方式, 0: 不交换, 1: 2字节交换, 3: 4字节交换; 无对应SDK的MV_PARAMTER 枚举参数。

[ADJUST_XXX]: 捕获时的信号调节设置。

VIDEO_STANDARD: 对应MV_PARAMTER的参数ADJUST_STANDARD。

VIDEO_SOURCE : 对应MV_PARAMTER的参数ADJUST_SOURCE。

VIDEO_CHANNEL : 对应MV_PARAMTER的参数ADJUST_CHANNEL。

VIDEO_BRIGHTNESS : 对应MV_PARAMTER的参数ADJUST_LUMINANCE。

VIDEO_HUE : 对应MV_PARAMTER的参数ADJUST_HUE。

VIDEO_SATURATION : 对应MV_PARAMTER的参数ADJUST_SATURATION。

VIDEO_CONTRAST : 对应 MV_PARAMTER 的参数 ADJUST_CONTRAST。

2.3 MV7130. INI

对应采集卡： V200、V120、V130、V160、V221、Moka_C20、Moka_C41、
Moka_M20、V410、S450、E410/E450、E411、E413、S260

配置文件名： Mv7130. ini

配置文件说明：

[DISPLAY]：该段定义了图像的显示参数。

DISP_TOP：图像显示在窗口的Y方向的起始偏移，对应SDK的MV_PARAMTER枚举参数DISP_TOP（默认值为：0；在窗口的Y方向从0开始显示）。

DISP_LEFT：图像显示在窗口的x方向的起始偏移，对应SDK的MV_PARAMTER枚举参数DISP_LEFT（默认值为：0；在窗口的X方向从0开始显示）。

DISP_HEIGHT：图像显示时的显示高度，对应SDK的MV_PARAMTER枚举参数DISP_HEIGHT（默认值为：576；在窗口中想要显示的图像高度为576）。

DISP_WIDTH：图像显示时的显示宽度，对应SDK的MV_PARAMTER枚举参数DISP_WIDTH（默认值为：768；在窗口中想要显示的图像高度为768）。

BACKGROUND：图像在显示时的窗口背景颜色，对应SDK的MV_PARAMTER枚举参数ADJUST_BACKCOLORKEY（默认值为：RGB(0, 0, 0x80)；取值范围：任意，但必须窗口背景颜色一致，当BUFFERTYPE=VIDOE0_MEMORY时有效）。

DISPFLIP：图像显示时上下翻转，对应SDK的MV_PARAMTER枚举参数DISP_FLIP。（默认值为：0；0 = 图像显示时不上下翻转；1 = 图像显示时上下翻转）。

[GARB]：捕获时的硬件相关的设置。

GARB_BITDESCRIBE：图像采集时的采集的图像的位深度，对应SDK的MV_PARAMTER枚举参数GARB_BITDESCRIBE（默认值为：3 (RGB32)；0 = Y8（灰度8位）；1 = RGB1555（16位）；2 = RGB24（24位）；3 = RGB8888

(32位)；4 = RGB8 (8位)；5 = RGB565 (16位)；6 = RGB5515 (16位)等等。注：16位在有些显卡上可能不能正常显示，此时建议更换一种)。

GARB_BUFFERTYPE: 选用的显示方式，同时也是采集时选用的缓冲类型，对应SDK的MV_PARAMTER枚举参数BUFFERTYPE (默认值为: 0 (DX模式)；0 = SYSTEM_MEMORY_DX (使用DirectX模式显示)；1 = SYSTEM_MEMORY_GDI (GDI模式显示)；2 = VIDEO_MEMORY (Overlay模式显示) 注：0-DirectX模式需要显卡支持 (需安装DirectX8或DirectX9)。1- GDI模式效率比较低 (资源占用率高))。

GARB_ISFILP : 图像采集时硬件的左右翻转，对应SDK的MV_PARAMTER枚举参数WORK_FLIP (默认值为: 0 (不左右翻转)；0 = 图像采集时不左右翻转；1 =图像采集时左右翻转)。

GARB_ISFIELD : 图像采集时确定采集时按场或按帧采集，对应SDK的MV_PARAMTER枚举参数WORK_FIELD (默认值为: 0 (按帧采集)；0 = 图像采集时按帧采集；1 = 图像采集时按场采集)。

GARB_ISUPDOWN: 图像采集时是否用硬件上下翻转，对应SDK的MV_PARAMTER枚举参数WORK_UPDOWN (默认值为: 0；0 = 图像采集时不上下翻转；1 =图像采集时上下翻转)。

GARB_XOFF, GARB_YOFF: 设置视频信号的X起始位置, Y起始位置。对应SDK的MV_PARAMTER枚举参数GARB_XOFF, GARB_YOFF (默认值为: 0；如采集的图像左边有黑边，可以调节GARB_XOFF。值应为2的整数倍；如采集的图像上边有黑边，可以调节GARB_YOFF)。

GARB_INPUT_HEIGHT, GARB_INPUT_WIDTH: 设置视频信号输入的高度, 宽度。对应MV_PARAMTER的参数GARB_IN_HEIGHT, GARB_IN_WIDTH。

GARB_HEIGHT, GARB_WIDTH: 设置视频信号输出的高度, 宽度 (即实际采集大小)。对应MV_PARAMTER的参数GARB_HEIGHT, GARB_WIDTH。

DISP_FIELD: 图像按场采集时的显示模式 (0:显示奇场, 1:显示偶场, 2:显示所有场)。

GARB_AUTOFIELD: 设置图像采集时是只采集奇场还是采集奇偶场。(默认值为: 4=采集奇偶场图像, 不管GARB_INPUT_HEIGHT和GARB_HEIGHT是多少；2 = 当采集图像的GARB_INPUT_HEIGHT和GARB_HEIGHT都小于288时，只采集奇场图像)

IMAGE_PROCESS: 图像按场采集时是否图像处理 (0: 否, 1 : 单场扩单帧, 2:保留)。

[ADJUST] : 捕获时的信号调节设置。

VIDEO_STANDARD: 对应MV_PARAMTER的参数ADJUST_STANDARD (默认值为: 0 (PAL制); 0 = PAL制, 1 = NTSC制)。

VIDEO_SOURCE : 对应MV_PARAMTER的参数ADJUST_SOURCE (默认值为: 0 (复合视频); 0 =复合视频, 1 = Svideo)。

VIDEO_CHANNEL : 对应MV_PARAMTER的参数ADJUST_CHANNEL (默认值为: 0; 不同板卡输入源数不同)。

VIDEO_BRIGHTNESS : 对应MV_PARAMTER的参数ADJUST_LUMINANCE (默认值为: 128; 取值范围: 0 - 255)。

VIDEO_HUE : 对应MV_PARAMTER的参数ADJUST_HUE默认值为: 128; 取值范围: 0 - 255)。

VIDEO_SATURATION : 对应MV_PARAMTER的参数ADJUST_SATURATION默认值为: 128; 取值范围: 0 - 255)。

VIDEO_CONTRAST : 对应MV_PARAMTER的参数ADJUST_CONTRAST默认值为: 128; 取值范围: 0 - 255)。

2.4 MVB.T.INI

对应采集卡：V110、V211、Moka_C10、Moka_C40、S100、V8T、X400/X800、S400/S420、

配置文件名：Mvbt.ini

配置文件说明：

[DISPLAY]：该段定义了图像的显示参数。

DISP_TOP：图像显示在窗口的Y方向的起始偏移，对应SDK的MV_PARAMTER枚举参数DISP_TOP（默认值为：0；在窗口的Y方向从0开始显示）。

DISP_LEFT：图像显示在窗口的x方向的起始偏移，对应SDK的MV_PARAMTER枚举参数DISP_LEFT（默认值为：0；在窗口的X方向从0开始显示）。

DISP_HEIGHT：图像显示时的显示高度，对应SDK的MV_PARAMTER枚举参数DISP_HEIGHT（默认值为：576；在窗口中想要显示的图像高度为576）。

DISP_WIDTH：图像显示时的显示宽度，对应SDK的MV_PARAMTER枚举参数DISP_WIDTH（默认值为：768；在窗口中想要显示的图像高度为768）。

BACKGROUND：图像在显示时的窗口背景颜色，对应SDK的MV_PARAMTER枚举参数ADJUST_BACKCOLORKEY（默认值为：RGB(0, 0, 0x80)；取值范围：任意，但必须窗口背景颜色一致，当BUFFERTYPE=VIDOE0_MEMORY时有效）。

DISPFLIP：图像显示时上下翻转，对应SDK的MV_PARAMTER枚举参数DISP_FLIP。（默认值为：0；0 = 图像显示时不上下翻转；1 = 图像显示时上下翻转）。

[GARB]：捕获时的硬件相关的设置。

GARB_BITDESCRIBE：图像采集时的采集的图像的位深度，对应SDK的MV_PARAMTER枚举参数GARB_BITDESCRIBE（默认值为：3 (RGB32)；0 = Y8（灰度8位）；1 = RGB1555（16位）；2 = RGB24（24位）；3 = RGB8888

(32位)；4 = RGB8 (8位)；5 = RGB565 (16位)；6 = RGB5515 (16位)等等。注：16位在有些显卡上可能不能正常显示，此时建议更换一种)。

GARB_BUFFERTYPE: 选用的显示方式，同时也是采集时选用的缓冲类型，对应SDK的MV_PARAMTER枚举参数BUFFERTYPE (默认值为: 0 (DX模式)；0 = SYSTEM_MEMORY_DX (使用DirectX模式显示)；1 = SYSTEM_MEMORY_GDI (GDI模式显示)；2 = VIDEO_MEMORY (Overlay模式显示) 注：0-DirectX模式需要显卡支持 (需安装DirectX8或DirectX9)。1- GDI模式效率比较低 (资源占用率高))。

GARB_ISFILP : 图像采集时硬件的左右翻转，对应SDK的MV_PARAMTER枚举参数WORK_FLIP (默认值为: 0 (不左右翻转)；0 = 图像采集时不左右翻转；1 = 图像采集时左右翻转)。

GARB_ISFIELD : 图像采集时确定采集时按场或按帧采集，对应SDK的MV_PARAMTER枚举参数WORK_FIELD (默认值为: 0 (按帧采集)；0 = 图像采集时按帧采集；1 = 图像采集时按场采集)。

GARB_ISUPDOWN: 图像采集时是否用硬件上下翻转，对应SDK的MV_PARAMTER枚举参数WORK_UPDOWN (默认值为: 0；0 = 图像采集时不上下翻转；1 = 图像采集时上下翻转)。

GARB_XOFF, GARB_YOFF: 设置视频信号的X起始位置, Y起始位置。对应SDK的MV_PARAMTER枚举参数GARB_XOFF, GARB_YOFF (默认值为: 0；如采集的图像左边有黑边，可以调节GARB_XOFF。值应为2的整数倍；如采集的图像上边有黑边，可以调节GARB_YOFF)。

GARB_INPUT_HEIGHT, GARB_INPUT_WIDTH: 设置视频信号输入的高度, 宽度。对应MV_PARAMTER的参数GARB_IN_HEIGHT, GARB_IN_WIDTH。

GARB_HEIGHT, GARB_WIDTH: 设置视频信号输出的高度, 宽度 (即实际采集大小)。对应MV_PARAMTER的参数GARB_HEIGHT, GARB_WIDTH。

DISP_FIELD: 图像按场采集时的显示模式 (0:显示奇场, 1:显示偶场, 2:显示所有场)。

GARB_AUTOFIELD: 设置图像采集时是只采集奇场还是采集奇偶场。(默认值为: 4=采集奇偶场图像, 不管GARB_INPUT_HEIGHT和GARB_HEIGHT是多少；2 = 当采集图像的GARB_INPUT_HEIGHT和GARB_HEIGHT都小于288时, 只采集奇场图像)

IMAGE_PROCESS: 图像按场采集时是否图像处理 (0: 否, 1 : 单场扩单帧, 2:保留)。

[ADJUST] : 捕获时的信号调节设置。

VIDEO_STANDARD: 对应MV_PARAMTER的参数ADJUST_STANDARD (默认值为: 0 (PAL制); 0 = PAL制, 1 = NTSC制)。

VIDEO_SOURCE : 对应MV_PARAMTER的参数ADJUST_SOURCE (默认值为: 0 (复合视频); 0 =复合视频, 1 = Svideo)。

VIDEO_CHANNEL : 对应MV_PARAMTER的参数ADJUST_CHANNEL (默认值为: 0; 不同板卡输入源数不同)。

VIDEO_BRIGHTNESS : 对应MV_PARAMTER的参数ADJUST_LUMINANCE (默认值为: 128; 取值范围: 0 - 255)。

VIDEO_HUE : 对应MV_PARAMTER的参数ADJUST_HUE默认值为: 128; 取值范围: 0 - 255)。

VIDEO_SATURATION : 对应MV_PARAMTER的参数ADJUST_SATURATION默认值为: 128; 取值范围: 0 - 255)。

VIDEO_CONTRAST : 对应MV_PARAMTER的参数ADJUST_CONTRAST默认值为: 128; 取值范围: 0 - 255)。

2.5 MVSQ. INI

对应采集卡：V400

配置文件名：Mvsq.ini

配置文件说明：

[DISPLAY]：该段定义了图像的显示参数。

DISP_TOP : 图像显示在窗口的Y方向的起始偏移，对应SDK的MV_PARAMTER枚举参数DISP_TOP。

DISP_LEFT : 图像显示在窗口的x方向的起始偏移，对应SDK的MV_PARAMTER枚举参数DISP_LEFT。

DISP_HEIGHT : 图像显示时的显示高度，对应SDK的MV_PARAMTER枚举参数DISP_HEIGHT。

DISP_WIDTH : 图像显示时的显示宽度，对应SDK的MV_PARAMTER枚举参数DISP_WIDTH。

BACKGROUND : 图像在显示时的窗口背景颜色，对应SDK的MV_PARAMTER枚举参数ADJUST_BACKCOLORKEY。//默认：RGB(0, 0, 0x80)

DISPFLIP : 图像显示时上下翻转，对应SDK的MV_PARAMTER枚举参数DISP_FLIP。

[GARB]：捕获时的硬件相关的设置。

GARB_BITDESCRIBE: 图像采集时的采集的图像的位深度，对应SDK的MV_PARAMTER枚举参数GARB_BITDESCRIBE。

GARB_BUFFERTYPE: 选用的显示方式，同时也是采集时选用的缓冲类型，对应SDK的MV_PARAMTER枚举参数BUFFERTYPE。

GARB_ISFILP : 图像采集时硬件的左右翻转，对应SDK的MV_PARAMTER枚举参数WORK_FLIP。

GARB_ISFIELD : 图像采集时确定采集时按场或按帧采集，对应SDK的MV_PARAMTER枚举参数WORK_FIELD。

GARB_ISUPDOWN : 图像采集时是否用硬件上下翻转, 对应SDK的MV_PARAMTER枚举参数WORK_UPDOWN。

GARB_XOFF, GARB_YOFF: 设置视频信号的X起始位置, Y起始位置。对应SDK的MV_PARAMTER枚举参数GARB_XOFF, GARB_YOFF。

GARB_INPUT_HEIGHT, GARB_INPUT_WIDTH: 设置视频信号输入的高度, 宽度。对应MV_PARAMTER的参数GARB_IN_HEIGHT, GARB_IN_WIDTH。

GARB_HEIGHT, GARB_WIDTH: 设置视频信号输出的高度, 宽度(即实际采集大小)。对应MV_PARAMTER的参数GARB_HEIGHT, GARB_WIDTH。

DISP_FIELD: 图像按场采集时的显示模式(0:显示奇场, 1:显示偶场, 2:显示所有场)。

GARB_AUTOFIELD: 图像自动按帧/场采集(2 : 是, 4 : 否)(保留)。

IMAGE_PROCESS: 图像按场采集时是否图像处理(0: 否, 1 : 单场扩单帧, 2:保留)。

[ADJUST] : 捕获时的信号调节设置。

VIDEO_STANDARD: 对应MV_PARAMTER的参数ADJUST_STANDARD。

VIDEO_SOURCE : 对应MV_PARAMTER的参数ADJUST_SOURCE。

VIDEO_CHANNEL : 对应MV_PARAMTER的参数ADJUST_CHANNEL。

VIDEO_BRIGHTNESS : 对应MV_PARAMTER的参数ADJUST_LUMINANCE。

VIDEO_HUE : 对应MV_PARAMTER的参数ADJUST_HUE。

VIDEO_SATURATION : 对应MV_PARAMTER的参数ADJUST_SATURATION。

VIDEO_CONTRAST : 对应MV_PARAMTER的参数ADJUST_CONTRAST。

2.6 MVVGA. INI

对应采集卡：Levin VGA

配置文件名：mvvga.ini

配置文件说明：

[DISPLAY]：该段定义了图像的显示参数。

 DISP_TOP ：图像显示在窗口的Y方向的起始偏移，对应SDK的MV_PARAMTER枚举参数DISP_TOP。

 DISP_LEFT ：图像显示在窗口的x方向的起始偏移，对应SDK的MV_PARAMTER枚举参数DISP_LEFT。

 DISP_HEIGHT ：图像显示时的显示高度，对应SDK的MV_PARAMTER枚举参数DISP_HEIGHT。

 DISP_WIDTH ：图像显示时的显示宽度，对应SDK的MV_PARAMTER枚举参数DISP_WIDTH。

 DISP_PRESENCE：图像是否显示，为1时图像采集后显示，为0时图像只采集不显示，对应MV_PARAMTER枚举参数DISP_PRESENCE。

[GARB]：捕获时的硬件相关的设置。

 IMAGE_XOffset, IMAGE_YOffset：设置视频信号的起始位置，对应SDK的MV_PARAMTER枚举参数GARB_XOFF, GARB_YOFF。

 IMAGE_YSize, IMAGE_XSize：设置视频信号的大小尺寸，对应SDK的MV_PARAMTER枚举参数GARB_HEIGHT, GARB_WIDTH。

 IMAGE_LineTotal：设置信号的行总数，对应SDK的MV_PARAMTER枚举参数GARB_WHOLEWIDTH, 和自动侦测的VIDEOSIGNAL枚举参数DIVIDER。

 IMAGE_Bit：设置采集图像的位深度，对应SDK的MV_PARAMTER枚举参数GARB_BITDESCRIBE。目前设置为32bit。

 IMAGE_VCOG、IMAGE_VCOR、IMAGE_BandWidth：和信号相关的卡的硬件设置，采集不同的VGA信号这几个参数有一定变化。

IMAGE_Interlace: 视频信号是否是隔行或逐行信号, 对应SDK的MV_PARAMTER枚举参数WORK_INTERLACE, 和自动侦测的VIDEOSIGNAL枚举参数ISINTERLACE, 常用的VGA信号都是逐行的。

IMAGE_UpDown: 图像采集时是否用硬件上下翻转, 对应SDK的MV_PARAMTER枚举参数WORK_UPDOWN。

[ADJUST]: 该段定义了VGA信号的R、G、B三路分量信号的亮度、对比度调节的参数。

ADJUST_R_LUM: 红路信号的亮度值, 对应SDK的MV_PARAMTER枚举参数ADJUST_R_LUM, 范围:0 - 255;

ADJUST_G_LUM: 绿路信号的亮度值, 对应SDK的MV_PARAMTER枚举参数ADJUST_G_LUM, 范围:0 - 255;

ADJUST_B_LUM: 蓝路信号的亮度值, 对应SDK的MV_PARAMTER枚举参数ADJUST_B_LUM, 范围:0 - 255;

ADJUST_R_COARSE: 红路分量上的对比度, 对应SDK的MV_PARAMTER枚举参数ADJUST_R_COARSE, 值为0 - 255;

ADJUST_G_COARSE: 绿路分量上的对比度, 对应SDK的MV_PARAMTER枚举参数ADJUST_G_COARSE, 值为0 - 255;

ADJUST_B_COARSE: 蓝路分量上的对比度, 对应SDK的MV_PARAMTER枚举参数ADJUST_B_COARSE, 值为0 - 255;