Zenkuman®

IEEE1394

Digital Camera Library

(ZCL-1)

Application Programming

Interface Specification

Technoscope Co., Ltd.

History

2005/09/12	First edition
2005/10/03	Description on ZCLOpen is changed. Way to use the standard value of ZCLCreateConvHandle is added. Incorrect description on ZCLSoftTrigger is corrected. Incorrect description on ZCLSetCallBack is corrected. A function is added. ZCLCameraInfo
2005/10/14	Description on ZCLImageCompleteWait is changed.
2005/11/02	Example in which ZCLSetCallBack is used is added. Incorrect description on ZCLGetLastError is corrected. Incorrect description on ZCLCameraInfo is corrected. Description on ZCLCameraInit is added. Description on ZCLSaveMem is corrected. Description on ZCLLoadMem is corrected. Description on ZCLIsoStart is corrected.
2006/01/20	ZCLColorConvSetBMPINFO function is added. ZCLSetPktSize function is added. ZCLCheckHitachi function is added. ZCLHMaskingOff function is added. ZCLHMaskingOn function is added. ZCLGetHMasking function is added.
2006/03/14	ZCLGetExtModeInfo function is added. ZCLImageCompleteWaitTimeOut function is added. The explanation of the structure definition is added. The explanation of the constant definition is added.

Contents

1. Outline

This specification describes the software of IEEE1394 Digital Camera Library “ZCL-1”. This is a functional specification for operating an IEEE1394 Digital Camera in the Windows operating environment of a DOS/V personal computer using “ZCL-1”.

2. Development Operation and Operating Environment

Application creation and operating environment in this specification

1) Computer	: IBM (DOS/V) personal computer	
Expansion bus	: Conforms to PCI revision 2.2.	
	One or more empty expansion bus	
OS	: Windows XP	
	: Windows 2000	
CPU	: Intel Pentium 233 MHz or higher	
Memory	: Mounting capacity of more than 64M byte	
Hard disk	: Mounting capacity of more than 1G byte	
2) IEEE1394 extended board	: Zenkuman PFW-85 (made by Technoscope)	
	: Zenkuman PFW-86 (made by Technoscope)	
3) Dedicated software	: ZCLDrv.sys	IEEE1394 interface card device driver
	: ZCL.dll	ZCL.API
4) Development environmental software	: Conforms to Microsoft Visual C++.	

Caution)

The person in charge of development requires C language-based software development experience in the Windows operating environment when using this software. Understanding the IEEE1394 Digital Camera Specification enables software to be developed smoothly.

3. Description of Functions

3.1 ZCL-1 Function List

3.1.1. Setting of system callback function

VOID ZCLSetCallBack (PVOID, Param
VOID(CALLBACK* SYSTEMFUNC)(STATUS_SYSTEMCODE , PVOID))

Argument

PVOID Param Value transferred to a callback function

VOID(CALLBACK*SYSTEMFUNC)(STATUS_SYSTEMCODE,PVOID)
Pointer to callback function

Return value

None

Interpretation

A callback function is set using this function when you want to receive the notification of the system state such as bus reset. To cancel the setting, set NULL to the second parameter.

VOID SYSTEMFUNC(STATUS_SYSTEMCODE Status, PVOID Context)

(ZCLAPI must not be called during processing of a callback function.)

○ Event for calling a callback function

• When bus reset is received

Status **STATUSZCL_BUSRESET**

Context Contents set to argument Param of this function

• When the sleep state is returned

Status **STATUSZCL_POWERUP**

Context Contents set to argument Param of this function

Example of use

- A callback function is set.

```
VOID CALLBACK SystemFunc(STATUS_SYSTEMCODE Status, PVOID Context )
{
    if( Status == STATUSZCL_BUSRESET )
    {
        //Processing of bus reset
    }
    if( Status == STATUSZCL_POWERUP )
```

```
        {
            //Processing of sleep cancellation
        }
    }

    if( ZCLSetCallBack( Param, SystemFunc ) )
    {
        //Succeeds in the setting of a callback function.
    }
    else
    {
        //Fails in the setting of a callback function.
    }
```

- The setting of a callback function is canceled.

  ZCLSetCallBack(NULL, NULL);

3.1.2. Acquisition of error code

STATUS_RTNCODE **ZCLGetLastError()**

Argument

None

Return value

Error code (See the status code list for details of the error code.)

Interpretation

The error code that was lastly generated is acquired.

See the status code list for details of the error code.

Example of use

- An error code is acquired.

```
STATUS_RTNCODE ErrorCode;

ErrorCode = ZCLGetLastError();
```

3.1.3. Acquisition of connection camera list

BOOL ZCLGetList(ZCL_LIST *List)

Argument

ZCL_LIST	*List	Stores the list of the currently connected cameras.	
		Pointer to a structure	

Return value

Success TRUE / Failure FALSE

Interpretation

The unique ID names, vender names, and model names of all cameras that are currently connected in a bus are acquired.

Specify the number of arrays in a list for List ->CameraCount.

The number of currently connected cameras is set for restoration when List->CameraCount is “0”.

Example of use

- The list of the connected cameras is acquired.

```
ZCL_LIST      *pList;
ULONG         Count;

Count = 0;
if( !ZCLGetList( (pZCL_LIST)&Count ) )
{
      //Fails in the acquisition of a list.
      return;
}
pList = (pZCL_LIST)malloc( sizeof(ZCL_LIST) + sizeof(pList->Info[ 0 ]) * Count );
pList->CameraCount = Count;
if( !ZCLGetList( pList ) )
{
      //Fails in the acquisition of a list.
      return;
}
//Succeeds in the acquisition of a list.
```

3.1.4. Start of camera use

BOOL ZCLOpen (UINT64 UID,
HCAMERA *hCamera)

Argument

UINT64	UID	Unique ID of the camera to be opened The camera that is not opened is automatically detected and opened when -1 is specified.
HCAMERA	*hCamera	Pointer for setting a camera handle

Return value

Success TRUE / Failure FALSE

Interpretation

The camera to be used is opened.

A camera handle value is used in each function of ZCL except the function on color conversion.

Be sure to open a camera using **ZCLOpen()** before using each function of ZCL.

During opening, the camera connected in an IEEE1394 bus is automatically recognized for processing by specifying -1 for UID.

Example of use

• A camera is opened.

```
HCAMERA     hCamera;
UINT64      UID;

UID = -1;
if( !ZCLOpen( UID, & hCamera ) )
{
    //Processing when a camera cannot be opened
    return;
}
```

3.1.5. Stop of camera use

BOOL ZCLClose(HCAMERA hCamera)

Argument

HCAMERA	hCamera	Handle value of the camera to be closed

Return value

Success TRUE / Failure FALSE

Interpretation

The camera opened using **ZCLOpen()** is closed.

Be sure to perform this processing when the use of a camera was stopped.

When the communication with the camera to be treated is done, the use of the camera is stopped after communication processing is stopped.

Be sure to execute this function when it is confirmed that a camera in the open state is not connected.

Example of use

• The opened camera is closed.

```
HCAMERA hCamera;

ZCLOpen( -1, &hCamera );

//Processing

ZCLClose( hCamera );
```

3.1.6. Entire stop of camera use

VOID ZCLAllClose()

Argument

None

Return value

None

Interpretation

The use of all opened cameras is stopped.

Example of use

- The use of all opened cameras is stopped.

  ZCLAllClose();

3.1.7. Acquisition of camera information

BOOL ZCLCameraInfo(HCAMERA hCamera,
ZCL_CAMERAINFO *pInfo,
ZCL_TRANSMITSPEED *pSpeed)

Argument

HCAMERA	hCamera	Handle value of the camera to be acquired
ZCL_CAMERAINFO	pInfo	Pointer for setting camera information
ZCL_TRANSMITSPEED	pSpeed	Pointer for setting communication rate information

Return value

Success TRUE / Failure FALSE

Interpretation

Information that a camera has is acquired.

The information is a unique ID name, vender name, model name, and maximum communication-possible rate.

NULL is set to each parameter when no information is required.

Example of use

• Information is acquired.

```
HCAMERA          hCamera;
ZCL_CAMERAINFO   Info;
ZCL_TRANSMITSPEED Speed;
if( ! ZCLCameraInfo( hCamera, &Info, &Speed ) )
{
    //Processing when a function failed
    return;
}
```

• A vender name is acquired.

```
ZCL_CAMERAINFO   Info;
if( ! ZCLCameraInfo( hCamera, &Info, NULL ) )
{
    //Processing when a function failed
    return;
}
```

3.1.8. Confirmation of camera connection

BOOL ZCLCamera(HCAMERA hCamera,
BOOL *Ari)

Argument

HCAMERA	hCamera	Handle value of the camera to be confirmed	
BOOL	*Ari	TRUE	Connected.
		FALSE	Not connected.

Return value

Success TRUE / Failure FALSE

Interpretation

It is confirmed whether a camera is in the connection state.

Be sure to execute **ZCLClose** when it is confirmed that a camera is not in the connection state.

Example of use

• The connection state of a camera is confirmed.

```
BOOL      Flag;
if( !ZCLCamera( hCamera, &Flag ) )
{
      //Processing when a function failed
      return;
}
if( !Flag )
{
      //Processing when a camera was put into the non-connection state
      return;
}
```

3.1.9. Initialization of camera

BOOL ZCLCameraInit(HCAMERA hCamera)

Argument

HCAMERA	hCamera	Handle value of the camera to be initialized

Return value

Success TRUE / Failure FALSE

Interpretation

A camera is initialized.

A camera is initialized using an initialization register that it has.

The initialized contents vary depending on each camera. For more details, see the camera manual.

Example of use

- A camera is put into the initialization state.

  ZCLCameraInit (hCamera);

3.1.10. Confirmation of camera mode

BOOL ZCLCheckCameraMode(HCAMERA hCamera,
ZCL_CAMERAMODE *Mode)

Argument

HCAMERA	hCamera	Handle value of the camera to be confirmed
ZCL_CAMERAMODE	*Mode	Mode value to be confirmed

Return value

Success TRUE / Failure FALSE

Interpretation

The camera mode that a camera has is checked.

A camera has the specified camera mode when the return value is TRUE.

When the return value is FALSE, a camera does not have the specified mode for **ZCLGetLastError()** and **STATUSZCL_NO_SUPPORT**.

Example of use

• The camera mode that a camera has is confirmed.

For standard mode

```
HCAMERA      hCamera;
ZCL_CAMERAMODE      mode;

mode.StdMode_Flag = TRUE;
mode.u.Std.Mode = ZCL_VGA_YUV1  //640 * 480 YUV411 mode
mode.u.Std.FrameRate = ZCL_Fps_30   //30 frames/s
if( !ZCLCheckCameraMode ( hCamera, &mode ) )
{
      //Processing when no mode exists
      return;
}
```

For extended mode (Format7)

```
HCAMERA      hCamera;
ZCL_CAMERAMODE      mode;
```

```
mode.StdMode_Flag = FALSE;
mode.u.Ext.Mode = ZCL_Mode_0
mode.u.Ext.ColorID = ZCL_MONO
if( !ZCLCheckCameraMode ( hCamera, &mode ) )
{
    //Processing when no mode exists
    return;
}
```

3.1.11. Setting of camera mode

BOOL ZCLSetCameraMode(HCAMERA hCamera,
ZCL_CAMERAMODE *Mode)

Argument

HCAMERA	hCamera	Handle value of the camera to be set
ZCL_CAMERAMODE	*Mode	Camera mode value to be set

Return value

Success TRUE / Failure FALSE

Interpretation

A camera is set to the specified camera mode.

Example of use

• A camera is set to the specified camera mode.

For standard mode

```
HCAMERA          hCamera;
ZCL_CAMERAMODE   mode;

mode.StdMode_Flag = TRUE;
mode.u.Std.Mode = ZCL_VGA_YUV1  //640 * 480 YUV411 mode
mode.u.Std.FrameRate = ZCL_Fps_30   //30 frames/s
if( !ZCLSetCameraMode ( hCamera, &mode ) )
{
    //Processing when no mode can be set
    return;
}
```

For extended mode (Format7)

```
HCAMERA          hCamera;
ZCL_CAMERAMODE   mode;

mode.StdMode_Flag = FALSE;
mode.u.Ext.Mode = ZCL_Mode_0
```

```
mode.u.Ext.ColorID = ZCL_MONO
if( !ZCLSetCameraMode ( hCamera, &mode ) )
{
    //Processing when no mode can be set
    return;
}
```

3.1.12. Acquisition of camera mode

BOOL ZCLNowCameraMode(HCAMERA hCamera,
ZCL_CAMERAMODE *Mode)

Argument

HCAMERA	hCamera	Handle value of the camera whose camera mode is acquired
ZCL_CAMERAMODE	*Mode	Pointer for setting the current camera mode value

Return value

Success TRUE / Failure FALSE

Interpretation

The current camera mode of a camera is acquired.

Example of use

• The current camera mode is acquired.

```
HCAMERA          hCamera;
ZCL_CAMERAMODE   mode;

if( !ZCLNowCameraMode( hCamera , &mode ) )
{
    //Processing when failure occurred
    return;
}
if( mode.StdMode_Flag )
{
    //Processing in the standard mode
}
else
{
    //Processing in the extended mode
}
```

3.1.13. Acquisition of enhancing mode information

BOOL ZCLGetExtModeInfo(HCAMERA hCamera,
ZCL_EXTMODEINFO *pInfo)

Argument

HCAMERA	hCamera	Handle value of the camera from which information is acquired
DWORD	Channel	Area where information is set

Return value

Success TRUE / Failure FALSE

Interpretation

Information on the enhancing mode that the camera has is acquired in the batch.

The information is number of mode, number of maximum pixels of each mode, and color coding ID..

Example of use

• Information on the enhancing mode that the camera has is acquired in the batch.

```
HCAMERA          hCamera;
ZCL_EXTMODEINFO  ExInfo;

if( !ZCLGetExtModeInfo ( hCamera,&ExInfo ) )
{
    //Processing when failure occurred
    return;
}
```

3.1.14. Saving of memory channel

BOOL **ZCLSaveMem(** **HCAMERA** **hCamera,**
DWORD Channel **)**

Argument

HCAMERA	hCamera	Handle value of a camera in which a memory channel is saved
DWORD	Channel	Channel number to be specified

Return value

Success TRUE / Failure FALSE

Interpretation

The current status of a camera is saved in a memory channel.

The current camera mode can also be saved depending on the camera.

The channel number is 1 to the maximum channel number. The maximum channel number is acquired using **ZCLCheckFeature**.

Depending on the camera, it may take time to complete saving.

Leave the interval of processing when performing the operation related to a camera after this function is executed.

(See the Users Manual of a camera for more information.)

Example of use

• The current status of a camera is saved in a memory channel.

```
HCAMERA         hCamera;


if( !ZCLSaveMem( hCamera, 2 ) )
{
        //Processing when failure occurred
        return;
}
```

3.1.15. Read of memory channel

BOOL **ZCLLoadMem(** **HCAMERA** **hCamera,**
DWORD Channel **)**

Argument

HCAMERA	hCamera	Handle value of a camera in which a memory channel is read
DWORD	Channel	Channel number to be specified

Return value

Success TRUE / Failure FALSE

Interpretation

The camera status and mode saved in the specified memory channel of a camera are read.

The channel number is 0 to the maximum channel number. The maximum channel number is acquired using **ZCLCheckFeature**.

"0" is the factory-setting value of a camera.

Example of use

• The saved camera status and mode are read.

```
HCAMERA         hCamera;

if( !ZCLLoadMem( hCamera, 0 ) )
{
    //Processing when failure occurred
    return;
}
```

3.1.16. Confirmation of camera mounting function

BOOL ZCLCheckFeature(HCAMERA hCamera,
ZCL_CHECKFEATURE *Feature)

Argument

HCAMERA	hCamera	Handle value of a camera whose mounting function is confirmed
ZCL_CHECKFEATURE	*Feature	Pointer for setting the information of a mounting function

Return value

Success TRUE / Failure FALSE

Interpretation

The function that a camera mounts is confirmed.

* The mounting function of a camera varies depending on the camera mode used. Therefore, confirm the camera mounting function again when changing the camera mode.

Example of use

• The mounting function of a camera is confirmed.

```
HCAMERA             hCamera;
ZCL_CHECKFEATURE    Feature;
Feature.Version = ZCLGetLibraryRevision();
Feature.FeatureID = ZCL_BRIGHTNESS;          //Brightness function
if( !ZCLCheckFeature( hCamera, &Feature ) )
{
    //Processing when failure occurred
    return;
}
if( Feature.PresenceFlag )
{
    //Processing when a function exists
}
```

3.1.17. Acquisition of camera parameter value

BOOL ZCLGetFeatureValue(HCAMERA hCamera,
ZCL_GETFEATUREVALUE *Value)

Argument

HCAMERA	hCamera	Handle value of a camera in which a camera parameter is acquired
ZCL_GETFEATUREVALUE	*Value	Pointer for setting a camera parameter value

Return value

Success TRUE / Failure FALSE

Interpretation

The value or state of a camera parameter are acquired.

Example of use

- The value and state of brightness are acquired.

```
HCAMERA             hCamera;
ZCL_GETFEATUREVALUE Value;

Value.Version = ZCLGetLibraryRevision();
Value.FeatureID = ZCL_BRIGHTNESS;
if( !ZCLGetFeatureValue( hCamera, &Value ) )
{
    //Processing when failure occurred
    return;
}
if( Value.u.Std.Auto_M )
{
    //Processing for automatic setting
}
```

- The value and state of WHITEBALANCE are acquired.

```
HCAMERA             hCamera;
ZCL_GETFEATUREVALUE Value;
```

```
ULONG               U, V;
Double              K;

Value.Version = ZCLGetLibraryRevision();
Value.FeatureID = ZCL_ WHITEBALANCE;
if( !ZCLGetFeatureValue( hCamera, &Value ) )
{
        //Processing when failure occurred
        return;
}
if( !Value.u.WhiteBalance. Abs )
{
        U = Value.u.WhiteBalance.UB_Value;
        V = Value.u.WhiteBalance.VR_Value
}
else
        K = Value.u.WhiteBalance.Abs_Value;
```

3.1.18. Setting of camera parameter value

BOOL ZCLSetFeatureValue (HCAMERA hCamera,
ZCL_SETFEATUREVALUE *Value)

Argument

HCAMERA	hCamera	Handle value of a camera in which a camera parameter is set
ZCL_SETFEATUREVALUE	*Value	Contents in which a camera parameter value is set

Return value

Success TRUE / Failure FALSE

Interpretation

A camera parameter is set to the specified value or state.

Example of use

• OnePush of WHITEBALANCE is executed.

```
HCAMERA             hCamera;
ZCL_SETFEATUREVALUE Value;

Value.Version = ZCLGetLibraryRevision();
Value.FeatureID = ZCL_ WHITEBALANCE;
Value.ReqID = ZCL_ONE_PUSH;
if( !ZCL_SetFeatureValue( hCamera, &Value ) )
{
    //Processing when failure occurred
    return;
}
```

• The automatic and manual modes of a shutter are set.

```
HCAMERA             hCamera;
ZCL_SETFEATUREVALUE Value;

Value.Version = ZCLGetLibraryRevision();
Value.FeatureID = ZCL_SHUTTER;
```

```
Value.ReqID = ZCL_AUTO;
if( !ZCL_SetFeatureValue( hCamera, &Value ) )
{
        //Processing when failure occurred
        return;
}

Value.Version = ZCLGetLibraryRevision();
Value.FeatureID = ZCL_SHUTTER;
Value.ReqID = ZCL_VALUE;
Value.u.Std.Value = 1000;
if( !ZCL_SetFeatureValue( hCamera, &Value ) )
{
        //Processing when failure occurred
        return;
}

Value.Version = ZCLGetLibraryRevision();
Value.FeatureID = ZCL_SHUTTER;
Value.ReqID = ZCL_ABSVALUE;
Value.u.Std.Abs_Value = 0.05;
if( !ZCL_SetFeatureValue( hCamera, &Value ) )
{
        //Processing when failure occurred
        return;
}
```

- A trigger mode is set and canceled.

```
HCAMERA             hCamera;
ZCL_SETFEATUREVALUE Value;

Value.Version = ZCLGetLibraryRevision();
Value.FeatureID = ZCL_TRIGGER;
Value.ReqID = ZCL_VALUE;
Value.u.Trigger.Polarity = 0;
Value.u.Trigger.Value = 0;
```

```
Value.u.Trigger.Source = ZCL_Trigger_Source0;
Value.u.Trigger.Mode = ZCL_Trigger_Mode0;
Value.u.Trigger.Parameter = 0;
if( !ZCL_SetFeatureValue( hCamera, &Value ) )
{
        //Processing when failure occurred
        return;
}

Value.Version = ZCLGetLibraryRevision();
Value.FeatureID = ZCL_TRIGGER;
Value.ReqID = ZCL_FEATURE_OFF;
if( !ZCL_SetFeatureValue( hCamera, &Value ) )
{
        //Processing when failure occurred
        return;
}
```

3.1.19. Securing of communication resources

BOOL ZCLIsoAlloc(HCAMERA hCamera)

Argument

HCAMERA hCamera Handle value of a camera in which communication resources are secured

Return value

Success TRUE / Failure FALSE

Interpretation

Necessary resources are secured in transferring an image to a camera.

The resources to be secured are the bandwidth and channel number during isochronous transmission.

Before executing this function, a camera mode must be set using **ZCLSetCameraMode()**.

The camera mode is set according to the default value (the current value set in a camera) when it is not set.

The current camera mode can be acquired using **ZCLNowCameraMode()**.

Communication resources may not be able to be secured when they have been secured in the set camera mode or using other cameras.

Be sure to execute **ZCLIsoRelease()** described later and then change the camera mode when a camera mode is changed using **ZCLSetCameraMode()** after this function is executed.

Example of use

• Communication resources are secured.

```
HCAMERA     hCamera;


if( !ZCLIsoAlloc( hCamera ) )
{
        //Processing when resources cannot be secured
        return;
}
```

3.1.20. Release of communication resources

BOOL ZCLIsoRelease(HCAMERA hCamera)

Argument

HCAMERA hCamera Handle value of a camera in which communication resources are released

Return value

Success TRUE / Failure FALSE

Interpretation

The communication resources secured using **ZCLIsoAlloc()** are released.

Example of use

• The communication resources of a camera are released.

```
HCAMERA hCamera;

ZCLIsoRelease( hCamera );
```

3.1.21. Start of communication

BOOL ZCLIsoStart(HCAMERA hCamera,
DWORD Frame)

Argument

HCAMERA	hCamera	Handle value of a camera in which communication is started
DWORD	Frame	The number of frames for communication is specified.
		1 : OneShot mode
		2~N : MultiShot mode
		0 : Free mode (usually during communication) or trigger mode

Return value

Success TRUE / Failure FALSE

Interpretation

The communication with the specified camera is started.

The communication resources of a camera must be secured using **ZCLIsoAlloc()** before executing this function.

Example of use

• Ordinary communication is performed.

```
HCAMERA     hCamera;

if( !ZCLIsoAlloc( hCamera ) )
{
        //Processing when failure occurred
        return;
}
if( !ZCLIsoStart( hCamera, 0 ) )
{
        //Processing when failure occurred
        return;
}
```

• Communication is done in the MultiShot mode. (10-frame reception)

```
ZCLIsoStart( hCamera, 10 );
```

• Communication is done in the OneShot mode.

ZCLIsoStart(hCamera, 1);

3.1.22. Stop of communication

BOOL ZCLIsoStop(HCAMERA hCamera)

Argument

HCAMERA hCamera	Handle value of a camera in which communication is stopped

Return value

Success TRUE / Failure FALSE

Interpretation

The communication with a camera is stopped.

Communication is stopped even for communication in the MultiShot mode.

Example of use

• The communication with a camera is stopped.

```
HCAMERA     hCamera;

ZCLIsoStop( hCamera );
```

3.1.23. Acquisition of image information

BOOL ZCLGetImageInfo(HCAMERA hCamera,
ZCL_GETIMAGEINFO *Info)

Argument

HCAMERA	hCamera	Handle value of a camera in which image information is acquired
ZCL_GETIMAGEINFO	*Info	Pointer of image information

Return value

Success TRUE / Failure FALSE

Interpretation

The image (image size and image data size) information of a camera is acquired.

Example of use

• Image information is acquired.

```
HCAMERA      hCamera;
ZCL_GETIMAGEINFO    Info;
WORD         Width, Height;
DWORD        Bufffer_Len, DataLength;
ZCL_COLORID  ColorID;

if( !ZCLGetImageInfo( hCamera ) )
{
    //Processing when failure occurred
    return;
}
Width = Info.Image.Width;
Height = Info.Image.Height;
ColorID = Info.Image.ColorID;
Bufffer_Len = Info.Image.Buffer;   //Image data length (including a padding byte)
DataLength = Info.Image.DataLength;   //Effective length of image data (not including a padding byte)
```

3.1.24. Setting of image information

BOOL ZCLSetImageInfo(HCAMERA hCamera,
ZCL_SETIMAGEINFO *Info)

Argument

HCAMERA	hCamera	Handle value of a camera in which image information is set
ZCL_SETIMAGEINFO	*Info	Image information value to be set

Return value

Success TRUE / Failure FALSE

Interpretation

The image (image size and image position) information of a camera is set.

This function is used to set the information of the image cut off when a camera mode is set to the expansion mode.

An error is restored when a camera mode is set to the standard mode.

Example of use

• The cut-off image in the expansion mode is set.

```
HCAMERA      hCamera;
ZCL_GETIMAGEINFO    GetInfo;
ZCL_SETIMAGEINFO    SetInfo;
ZCL_COLORID         ColorID;
DWORD    Buffer_Len, DataLength;

if( !ZCLGetImageInfo( hCamera, &GetInfo ) )
{
        //Processing when failure occurred
        return;
}
if( GetInfo.StdMode_Flag )
{
        //Processing in the standard mode
        return;
}
```

```
SetInfo.PosX = GetInfo.Ext.UnitPosX * 2;
SetInfo.PosY = GetInfo.Ext.UnitPosY * 5;
SetInfo.Width = GetInfo.Ext.UnitSizeX * 3;
SetInfo.Height = GetInfo.Ext.UnitSizeY * 4;
SetInfo.MaxSize_Flag = false;        //Packet length is set to the minimum value.
if( !ZCLSetImageInfo( hCamera, &SetInfo ) )
{
        //Processing when failure occurred
        return;
}
if( !ZCLGetImageInfo( hCamera, &GetInfo ) )
{
        //Processing when failure occurred
        return;
}
Buffer_Len = GetInfo.Image.Buffer; //Image data length (including a padding byte)
DataLength = GetInfo.Image.DataLength; //Effective length of image data (not including a padding byte)
ColorID = GetInfo.Image.ColorID;
```

3.1.25. Setting of packet size

BOOL ZCLSetPktSize (HCAMERA hCamera,
DWORD Number,
DWORD *pPktLen)

Argument

HCAMERA	hCamera	Handle value of a camera in which image information is set
DWORD	Number	Multiple (>2) of the packet length used as standard
DWORD	*pPktLen	Packet length to be set

Return value

Success TRUE / Failure FALSE

Interpretation

This function sets the information of the image cut off when a camera mode is set to the expansion mode and then sets the packet size.

Packet length is specified for the multiple length specified with the minimum packet length, which a camera calculated according to the specified image size, as reference.

Example of use

• Packet size is specified after the cut-off image in the expansion mode is set.

```
HCAMERA      hCamera;
ZCL_GETIMAGEINFO    GetInfo;
ZCL_SETIMAGEINFO    SetInfo;
ZCL_COLORID         ColorID;
DWORD    Buffer_Len, DataLength, PktLen;

if( !ZCLGetImageInfo( hCamera, &GetInfo ) )
{
        //Processing when failure occurred
        return;
}
if( GetInfo.StdMode_Flag )
{
        //Processing in the standard mode
```

```
        return;
}
SetInfo.PosX = GetInfo.Ext.UnitPosX * 2;
SetInfo.PosY = GetInfo.Ext.UnitPosY * 5;
SetInfo.Width = GetInfo.Ext.UnitSizeX * 3;
SetInfo.Height = GetInfo.Ext.UnitSizeY * 4;
SetInfo.MaxSize_Flag = false;       //Packet length is set to the minimum value.
if( !ZCLSetImageInfo( hCamera, &SetInfo ) )
{
        //Processing when failure occurred
        return;
}
if( !ZCLSetPktSize ( hCamera, 3, &PktLen ) )
{
        //Processing when failure occurred
        return;
}
if( !ZCLGetImageInfo( hCamera, &GetInfo ) )
{
        //Processing when failure occurred
        return;
}
Buffer_Len = GetInfo.Image.Buffer;      //Image data length (including a padding byte)
DataLength = GetInfo.Image.DataLength; //Effective length of image data (not including a padding byte)
ColorID = GetInfo.Image.ColorID;
```

3.1.26. Acquisition request of image data

BOOL ZCLImageReq(HCAMERA hCamera,
BYTE *pBuf,
DWORD Len)

Argument

HCAMERA	hcamera	Handle value of a camera in which an image is acquired
BYTE	*pBuf	Pointer of the place where image data is stored
DWORD	Len	Image data buffer length

Return value

Success TRUE / Failure FALSE

Interpretation

Image data is acquired from a camera.

Image data can be acquired proportionally to the number of execution times by executing this function before starting the communication with a camera.

The next frame data is acquired in the course of a frame when this function is executed during communication with a camera.

This function requests the reception of image data. It is immediately restored without waiting that the reception of image data is completed.

Using **ZCLImageCompleteWait** or **ZCLImageComplateWaitTimeOut** function, confirm that the reception of image data is completed.

Example of use

• Image data is acquired after communication is started.

```
HCAMERA             hCamera;
BYTE                *pData;
ZCL_GETIMAGEINFO    Info;

ZCLGetImageInfo( hCamera, &Info );
pData = ( BYTE *)malloc( Info.Image.Buffer );
ZCLIsoStart( hCamera, 0 );
ZCLImageReq( hCamera , pData, Info.Image.Buffer );
ZCLImageCompleteWait( hCamera, pData, 0, 0, 0 );
```

```
ZCLIsoStop( hCamera );
free( pData );
```

• Before starting communication, data is acquired after the acquisition of data is set.

```
HCAMERA             hCamera;
BYTE                *pData[ 3 ];
ZCL_GETIMAGEINFO    Info;
int         idx;

ZCLGetImageInfo( hCamera, &Info );
for( idx = 0; idx < 3; idx++ )
{
    pData[ idx ] = ( BYTE *)malloc( Info.Image.Buffer );
    ZCLImageReq( hCamera , pData[ idx ], Info.Image.Buffer );
}
ZCLIsoStart( hCamera, 0 );
for( idx = 0; idx < 3; idx++ )
    ZCLImageCompleteWait( hCamera, pData[ idx ], 0, 0, 0 );

ZCLIsoStop( hCamera );
for( idx = 0; idx < 3; idx++ )
    free( pData[ idx ] );
```

3.1.27. Acquisition completion wait of image data

```
BOOL    ZCLImageCompleteWait( HCAMERA              hCamera,
                              BYTE                 *pBuf,
                              ZCL_TRANSMITSPEED    *pSpeed,
                              DWORD                *pCycleTime,
                              DWORD                *pCycleCount )
```

Argument

HCAMERA	hCamera	Handle value of a camera by which the completion of data reception is confirmed
BYTE	*pBuf	Pointer of the place where image data is stored
ZCL_TRANSMITSPEED	*pSpeed	Pointer in which a communication rate is stored
DWORD	*pCycleTime	Pointer in which bus time is stored
DWORD	*pCycleCount	Pointer in which a bus count is stored

Return value

Success TRUE / Failure FALSE

Interpretation

It is confirmed by a camera whether the reception of image data was completed.

This function is not restored until the reception completion of image data is confirmed by the specified buffer.

The relevant value is not set when the pointer of the argument such as a communication rate, bus time, and bus count is not set (when NULL is specified).

The bus time is a repetend of 0 to 127. The unit used indicates a second.

The bus count is a repetend of 0 to 7999. It is indicated in units of 125μsec.

Example of use

• The completion of image data is confirmed. The information on communication rate is not used.

```
HCAMERA             hCamera;
BYTE                *pData;
ZCL_GETIMAGEINFO    Info;

ZCLGetImageInfo( hCamera, &Info );
pData = ( BYTE *)malloc( Info.Image.Buffer );
```

```
ZCLIsoStart( hCamera, 0 );
ZCLImageReq( hCamera , pData, Info.Image.Buffer );
ZCLImageCompleteWait( hCamera, pData, NULL, NULL, NULL );
ZCLIsoStop( hCamera );
free( pData );
```

• The completion of image data is confirmed. The information on communication rate is used.

```
HCAMERA             hCamera;
BYTE                *pData;
ZCL_GETIMAGEINFO    Info;
ZCL_TRANSMITSPEED   Speed;
DOWRD               CycleTime, CycleCount;

ZCLGetImageInfo( hCamera, &Info );
pData = ( BYTE *)malloc( Info.Image.Buffer );
ZCLIsoStart( hCamera, 0 );
ZCLImageReq( hCamera , pData, Info.Image.Buffer );
ZCLImageCompleteWait( hCamera, pData, &Speed, &CycleTime, &CycleCount );
ZCLIsoStop( hCamera );
 free( pData );
```

3.1.28. Acquisition completion wait of image data (with time-out)

BOOL ZCLImageCompleteWaitTimeOut(HCAMERA hCamera,
BYTE *pBuf,
ZCL_TRANSMITSPEED *pSpeed,
DWORD *pCycleTime,
DWORD *pCycleCount,
DWORD Time)

Argument

HCAMERA	hCamera	Handle value of a camera by which the completion of data reception is confirmed
BYTE	*pBuf	Pointer of the place where image data is stored
ZCL_TRANSMITSPEED	*pSpeed	Pointer in which a communication rate is stored
DWORD	*pCycleTime	Pointer in which bus time is stored
DWORD	*pCycleCount	Pointer in which a bus count is stored
DWORD	Time	Waiting time, the unit is a millisecond.

Return value

Success TRUE / Failure FALSE

Interpretation

This function is the same function as **ZCLImageCompleteWait** function, and can specify waiting time.

When "0" is specified at waiting time, completion is confirmed , and it returns at once.

When "-1" is specified at waiting time, it becomes the same operation as **ZCLImageCompleteWait** function.

Example of use

• The completion of image data is waited for at 100 millisecond. The information on communication rate is not used.

```
HCAMERA             hCamera;
BYTE                *pData;
ZCL_GETIMAGEINFO    Info;

ZCLGetImageInfo( hCamera, &Info );
pData = ( BYTE *)malloc( Info.Image.Buffer );
```

```
ZCLIsoStart( hCamera, 0 );
ZCLImageReq( hCamera , pData, Info.Image.Buffer );
if( !ZCLImageCompleteWaitTimeOut( hCamera, pData, NULL, NULL, NULL,100) )
{
        //Processing of time-out
}
else
{
        //Processing of image data acquisition
}
ZCLIsoStop( hCamera );
free( pData );
```

3.1.29. Acquisition cancellation of image data

BOOL ZCLAbortImageReqAll(HCAMERA hCamera)

Argument

HCAMERA	hcamera	Handle value of a camera in which the acquisition of image data is Canceled

Return value

Success TRUE / Failure FALSE

Interpretation

An image acquisition request command executed using **ZCLImageReq()** and a command put into the completion wait state using **ZCLImageCompleteWait()** are canceled.

Example of use

• The image data acquisition command to be requested is canceled.

```
HCAMERA             hCamera;
BYTE                *pData[ 3 ];
ZCL_GETIMAGEINFO    Info;
int         idx;

ZCLGetImageInfo( hCamera, &Info );
for( idx = 0; idx < 3; idx++ )
{
    pData[ idx ] = ( BYTE *)malloc( Info.Image.Buffer );
    ZCLImageReq( hCamera , pData[ idx ], Info.Image.Buffer );
}
ZCLAbortImageReqAll( hCamera );
for( idx = 0; idx < 3; idx++ )
    free( pData[ idx ] );
```

3.1.30. Execution of soft trigger

BOOL ZCLSoftTrigger(HCAMERA hCamera
BOOL OnOff)

Argument

HCAMERA	hcamera	Handle value of a camera in which soft trigger is executed
BOOL	OnOff	On and Off are specified.
		FALSE Specifies Off.
		TRUE Specifies On.

Return value

Success TRUE / Failure FALSE

Interpretation

A soft trigger command is issued.

Please execute Off about the soft trigger command after executing On.

However, when trigger mode 0 is set, the library executes Off by the automatic operation.

Example of use

• Soft trigger is issued.

```
HCAMERA          hCamera;

ZCLSoftTriger( hCamera, TRUE );
Sleep( 100 );
ZCLSoftTriger( hCamera, FALSE );
```

3.1.31. Read of camera control register

BOOL ZCLGetRegister(HCAMERA hCamera,
ZCL_REGISTER *pReg)

Argument

HCAMERA	hcamera	Handle value of a camera in which a register is read
ZCL_REGISTER	*pReg	A register is read. The offset, length, and storage pointer are specified.

Return value

Success TRUE / Failure FALSE

Interpretation

A camera control register is read.

A base address uses the value defined using ConfigROM.

Read length is specified in units of four bytes.

The value to be read corresponds to the contents displayed in DWORD.

The read contents are 0x80020040 when the register value of a camera is 0x80 0x02 0x00 0x40.

Example of use

- A BASIC_FUNC_INQ register is read.

```
HCAMERA          hCamera;
ZCL_REGISTER     Reg;
DWORD            RegData;

Reg.Offset = 0x400;
Reg.Size = 4;
Reg.Value = &RegData;
ZCLGetRegister( hCamera, &Reg );
// Read in RegData.
```

3.1.32. Read of register

BOOL ZCLGetExtRegister(HCAMERA hCamera,
ZCL_REGISTER *pReg)

Argument

HCAMERA	hcamera	Handle value of a camera in which a register is read
ZCL_REGISTER	*pReg	A register is read. The offset, length, and storage pointer are specified.

Return value

Success TRUE / Failure FALSE

Interpretation

The register of the specified offset address is read.

A base address is 0x0ffff f000 0000.

Read length is specified in units of four bytes.

The value to be read corresponds to the contents displayed in DWORD.

The read contents are 0x80020040 when the register value of a camera is 0x80 0x02 0x00 0x40.

Example of use

- A Format7 Mode0 Max_IMAGE_SIZE_INQ register is read.

```
HCAMERA         hCamera;
ZCL_REGISTER    Reg;
DWORD           Data;

Reg.Offset = 0x02e0;
Reg.Size = 4;
Reg.Value = &Data;
ZCLGetRegister( hCamera, &Reg );
Reg.Offset = Data << 2;
ZCLGetExtRegister( hCamera, &Reg );
// Read in Data.
```

3.1.33. Write of camera control register

BOOL ZCLSetRegister(HCAMERA hCamera,
ZCL_REGISTER *pReg)

Argument

HCAMERA	hcamera	Handle value of a camera in which a register is written
ZCL_REGISTER	*pReg	A register is written. The offset, length, and data pointer are specified.

Return value

Success TRUE / Failure FALSE

Interpretation

A camera control register is written.

A base address uses the value defined using ConfigROM.

Write length is specified in units of four bytes.

The value to be written corresponds to the contents displayed in DWORD.

The register value of a camera is 0x80 0x02 0x00 0x40 when the write contents are 0x80020040.

Example of use

- The start of image transfer is set to a camera.

```
HCAMERA         hCamera;
ZCL_REGISTER    Reg;
DWORD           RegData;

Reg.Offset = 0x614;
Reg.Size = 4;
Reg.Value = &RegData;
RegData = 0x80000000;
ZCLSetRegister( hCamera, &Reg );
```

3.1.34. Write of register

BOOL ZCLSetExtRegister(HCAMERA hCamera,
ZCL_REGISTER *pReg)

Argument

HCAMERA	hcamera	Handle value of a camera in which a register is written
ZCL_REGISTER	*pReg	A register is written. The offset, length, and data pointer are specified.

Return value

Success TRUE / Failure FALSE

Interpretation

The register of the specified offset address is written.

A base address is 0x0ffff f000 0000.

Write length is specified in units of four bytes.

The value to be written corresponds to the contents displayed in DWORD.

The register value of a camera is 0x80 0x02 0x00 0x40 when the write contents are 0x80020040.

Example of use

• MONO16 is specified in a Format7 Mode0 COLOR_CODING_ID register.

```
HCAMERA         hCamera;
ZCL_REGISTER    Reg;
DWORD           Data;

Reg.Offset = 0x02e0;
Reg.Size = 4;
Reg.Value = &Data;
ZCLGetRegister( hCamera, &Reg );
Reg.Offset = ( Data << 2 ) + 0x010;
Data = 5 << 24;
ZCLSetExtRegister( hCamera, &Reg );
```

3.1.35. Acquisition of library revision

DWORD ZCLGetLibraryRevision()

Argument

None

Return value

Library revision number

Interpretation

The revision number of a library is acquired.

In version 1.20, the revision number is 120.

The revision number is changed and updated when addition (or change) occurs in the mounted function.

Example of use

• A version is acquired.

```
if( ZCLGetLibraryRevision() == 120 )
{
    //For a version of 1.20
}
else
{
    //For versions other than 1.20
}
```

3.1.36. Reinitialization of isochronous resources

BOOL ZCLReset(void)

Argument

None

Return value

Success TRUE / Failure FALSE

Interpretation

Isochronous resources are reinitialized.

* Usually, do not use this function.

Example of use

ZCLReset();

3.1.37. Creation of color conversion table

BOOL ZCLCreateConvHandle (HCTBL *hTbl,
ZCL_CONVERTMODE Mode,
ZCL_SHIFTID Shift,
ZCL_COLORVALUE *pValue)

Argument

HCTBL	*hTbl	Pointer for setting a color conversion handle
ZCL_CONVERTMODE	Mode	A conversion mode is set.
ZCL_SHIFTID	Shift	A bit shift is specified.
ZCL_COLORVALUE	*pValue	A color conversion coefficient is specified.

Return value

Success TRUE / Failure FALSE

Interpretation

The resources for converting in the mode in which the image data sent from a camera was specified are created.

32-bit, 24-bit, 16-bit, 15-bit, and color filter (RAW8 and RAW16) color conversion modes are specified for conversion mode.

The conversion mode is fixed to 32-bit color when a color filter is specified.

For the specification of a bit shift, the bit to be selected is specified when image data is eight-bit gradation and above.

For a color conversion coefficient, the coefficient during color conversion from YUV information to RGB information is specified. The coefficient is specified using the expression below.

$R = Y + pValue\text{->}\boldsymbol{a} \times V$

$G = Y - pValue\text{->}\boldsymbol{b} \times V - pValue\text{->}\boldsymbol{c} \times U$

$B = Y + pValue\text{->}\boldsymbol{d} \times U$

When no coefficient is specified (NULL is set to pValue), color conversion is done as standard by the expression below.

pValue->a = 1.402

pValue->b = 0.711

pValue->c = 0.343

pValue->d = 1.772

Example of use

- A color conversion coefficient performs 24-bit color conversion as a standard value.

```
HCTBL          hTbl;

ZCLCreateConvHandle ( & hTbl, ZCL_C24bit, ZCL_SFT0, NULL );
```

- A color conversion coefficient is set to a = 1.5, b = 0.8, c = 0.4, and d =1.9 so as to perform 24-bit color conversion.

```
HCTBL          hTbl;
ZCL_COLORVALUE          Value;

Value.a = 1.5;
Value.b = 0.8;
Value.c = 0.4;
Value.d = 1.9;
ZCLCreateConvHandle ( & hTbl, ZCL_C24bit, ZCL_SFT0, &Value );
```

3.1.38. Opening of color conversion table

VOID ZCLCloseConvHandle(HCTBL hTbl)

Argument

HCTBL	hTbl	Handle value of the color conversion table to be opened

Return value

None

Interpretation

The table created using **ZCLCreateConvHandle()** is opened.

Example of use

- A color conversion table is opened.

```
HCTBL hTbl;

ZCLCloseConvHandle( hTbl );
```

3.1.39. Entire opening of color conversion table

VOID ZCLCloseAllConvHandle( )

Argument

None

Return value

None

Interpretation

All tables created using **ZCLCreateConvHandle()** are opened.

Example of use

• All color conversion tables are opened.

```
ZCLCloseAllConvHandle( );
```

3.1.40. Execution of color conversion

BOOL ZCLColorConvExec (HCTBL hTbl,
DWORD Width,
DWORD Height,
ZCL_COLORMODE *pMode,
BYTE *SrcBuf,
BYTE *DstBuf)

Argument

HCTBL	hTbl	Handle value of the color conversion table to be opened
DWORD	Width	Width of image
DWORD	Height	Height of image
ZCL_COLORMODE	*pMode	Conversion information
		Color identification of original data
		Identification of color filter
		Data array after conversion
BYTE	*SrcBuf	Buffer pointer in which conversion source image data is stored
BYTE	*DstBuf	Buffer pointer in which the converted image data is stored

Return value

Success TRUE / Failure FALSE

Interpretation

Color conversion is executed.

When **ZCL_BMPmode** is specified by specifying the data array after conversion, the number of bytes in one line is automatically adjusted to a four-byte boundary if it is not a four-byte boundary.

Example of use

• The array after conversion is reversed at top and bottom for color conversion.

```
BYTE        Data[ 20 * 20 ];
BYTE        BMP[ 20 * 20 * 3 ];
ZCL_COLORMODE Mode;
HCTBL       hTbl;

Mode.ColorID = ZCL_YUV422;
```

```
Mode. Cfilter = 0;
Mode. StoreMode = ZCL_BMPmode;
ZCLColorConvExec( hTbl, 20, 20, &Mode, &Data[ 0 ], &BMP[ 0 ] );
```

• Color conversion is executed.

```
BYTE        Data[ 20 * 20 ];
BYTE        BMP[ 20 * 20 * 3 ];
ZCL_COLORMODE  Mode;
HCTBL       hTbl;

Mode.ColorID = ZCL_YUV422;
Mode. Cfilter = 0;
Mode. StoreMode = ZCL_MEMmode;
ZCLColorConvExec( hTbl, 20, 20, &Mode, &Data[ 0 ], &BMP[ 0 ] );
```

3.1.41. Setting of BITMAPINFO information

BOOL ZCLColorConvSetBMPINFO (HCTBL hTbl,
DWORD Width,
DWORD Height,
BITMAPINFO *pBmpInfo)

Argument

HCTBL	hTbl	Handle value of the color conversion table to be opened
DWORD	Width	Width of image
DWORD	Height	Height of image
BITMAPINFO	*pBmpInfo	BMPINFO information table

Return value

Success TRUE / Failure FALSE

Interpretation

The BITMAPINFO information when converting into a BMP format during color conversion is set.

Example of use

• BITMAPINFO information is set.

```
BITMAPINFO    BmpInfo;
HCTBL    hTbl;

ZCLColorConvSetBMPINFO ( hTbl, 20, 20, & BmpInfo );
```

3.1.42. Confirmation of Hitachi Kokusai camera model

BOOL ZCLCheckHitachi (HCAMERA hCamera,)

Argument

HCAMERA	hCamera	Handle value of a camera in which a Hitachi Kokusai camera model is confirmed

Return value

Success TRUE / Failure FALSE

Interpretation

This function must be called once when the vender unique function of a camera made by Hitachi Kokusai is used.

The functions below can be used after this function is called.

ZCLHMaskingOn()

ZCLHMaskingOff()

ZCLGetHMasking()

Example of use

• The use of a new expanded function made by Hitachi Kokusai is declared.

```
HCAMERA          hCamera;

ZCLCheckHitachi( hCamera );
```

3.1.43. Setting of masking

BOOL ZCLHMaskingOn(HCAMERA hCamera
ZCL_HMASKING *pMasking)

Argument

HCAMERA	hCamera	Handle value of a camera in which masking is set
ZCL_HMASKING	pMasking	Masking value to be set

Return value

Success TRUE / Failure FALSE

Interpretation

A masking value is set for each color.

Example of use

• A masking value is set.

HCAMERA hCamera;
ZCL_HMASKING Masking;

Masking.Saturaton.R = 0x80;
Masking.Saturaton.G = 0x80;
Masking.Saturaton.B = 0x80;
Masking.Saturaton.Y = 0x80;
Masking.Saturaton.C = 0x80;
Masking.Saturaton.M = 0x80;
Masking. Hue.R = 0x80;
Masking. Hue.G = 0x80;
Masking. Hue.B = 0x80;
Masking. Hue.Y = 0x80;
Masking. Hue.C = 0x80;
Masking. Hue.M = 0x80;
ZCLHMaskingOn(hCamera, &Masking);

3.1.44. Release of masking setting

BOOL ZCLHMaskingOff(HCAMERA hCamera)

Argument

HCAMERA	hCamera	Handle value of a camera in which the setting of masking is released

Return value

Success TRUE / Failure FALSE

Interpretation

A masking function is released.

Example of use

- A masking function is released.

```
HCAMERA          hCamera;

ZCLHMaskingOff ( hCamera );
```

3.1.45. Acquisition of masking value

BOOL ZCLGetHMasking(HCAMERA hCamera,
BOOL *pOnOff,
ZCL_HMASKING *pMasking)

Argument

HCAMERA	hCamera	Handle value of a camera in which a masking value is acquired
BOOL	pOnOff	Masking function state 0 Invalid 1 Valid
ZCL_HMASKING	pMasking	Current setting value

Return value

Success TRUE / Failure FALSE

Interpretation

The current setting value of masking is acquired.

Example of use

• A masking value is acquired.

```
HCAMERA          hCamera;
BOOL             OnOff;
ZCL_HMASKING     Masking;

ZCLGetHMasking ( hCamera, &OnOff, &Masking );
```

3.2 Status codes

STATUSZCL_NO_ERROR	No error
STATUSZCL_COMPLETE	Success in processing
STATUSZCL_PARAMETER_ERROR	A parameter is illegal.
STATUSZCL_BUFFER_SHORT	A buffer is short.
STATUSZCL_OPEN_ERROR	Failure in the opening of a camera
STATUSZCL_OPENED	A camera has been already opened.
STATUSZCL_CANNOT_FOUND	A camera cannot be found.
STATUSZCL_NO_OPEN	A camera is not opened.
STATUSZCL_COMMUNICATE_ERROR	Failure in communication
STATUSZCL_DATA_INACCURACY	The acquired data is inaccurate.
STATUSZCL_NO_SUPPORT	No function is mounted.
STATUSZCL_VMODE_ERROR	Camera parameter setting error
STATUSZCL_FEATURE_ERROR	Camera feature control error
STATUSZCL_VALUE_ERROR	Camera parameter setting error
STATUSZCL_SELFCLEAR_ERROR	A self-clear flag is not cleared.
STATUSZCL_IMAGE_ERROR	Image size setting error
STATUSZCL_RESOURCE_ERROR	Requested. during securing of isochronous resources
STATUSZCL_NOTRESOURCE_ERROR	Isochronous resources are not secured.
STATUSZCL_ALLOCATE_ERROR	Failure in the securing of isochronous resources
STATUSZCL_STARTED_ERROR	The start state has been already reached.
STATUSZCL_NOTSTART_ERROR	The start state is not reached.
STATUSZCL_REQUEST_ERROR	Failure in image request
STATUSZCL_REQUEST_TIMEOUT	Time-out in image request
STATUSZCL_SOFTTRIGGER_BUSY	During soft trigger execution
STATUSZCL_MULTISLOPE_ERROR	Multi-slope setting error
STATUSZCL_UNDEF_ERROR	Other errors

3.3 Constant

Mode of standard format

```
typedef enum
{
    ZCL_QQVGA               = 0,    // 160 x 120 YUV(4:4:4)
    ZCL_QVGA,                       // 320 x 240 YUV(4:2:2)
    ZCL_VGA_YUV1,                   // 640 x 480 YUV(4:1:1)
    ZCL_VGA_YUV2,                   // 640 x 480 YUV(4:2:2)
    ZCL_VGA_RGB,                    // 640 x 480 RGB
    ZCL_VGA_MONO,                   // 640 x 480 Mono
    ZCL_VGA_MONO16,                 // 640 x 480 Mono16
    ZCL_SVGA_YUV,                   // 800 x 600 YUV(4:2:2)
    ZCL_SVGA_RGB,                   // 800 x 600 RGB
    ZCL_SVGA_MONO,                  // 800 x 600 MONO
    ZCL_SVGA_MONO16,                // 800 x 600 MONO16
    ZCL_XGA_YUV,                    // 1024 x 768 YUV(4:2:2)
    ZCL_XGA_RGB,                    // 1024 x 768 RGB
    ZCL_XGA_MONO,                   // 1024 x 768 MONO
    ZCL_XGA_MONO16,                 // 1024 x 768 MONO16
    ZCL_SXGA_YUV,                   // 1280 x 960 YUV(4:2:2)
    ZCL_SXGA_RGB,                   // 1280 x 960 RGB
    ZCL_SXGA_MONO,                  // 1280 x 960 MONO
    ZCL_SXGA_MONO16,                // 1280 x 960 MONO16
    ZCL_UXGA_YUV,                   // 1600 x 1200 YUV(4:2:2)
    ZCL_UXGA_RGB,                   // 1600 x 1200 RGB
    ZCL_UXGA_MONO,                  // 1600 x 1200 MONO
    ZCL_UXGA_MONO16,                // 1600 x 1200 MONO16
} ZCL_STDMODE;
```

Frame rate of standard format

```
typedef enum
{
    ZCL_Fps_1875            = 0,    // 1.875fps
    ZCL_Fps_375,                    // 3.75fps
    ZCL_Fps_75,                     // 7.5fps
    ZCL_Fps_15,                     // 15fps
    ZCL_Fps_30,                     // 30fps
    ZCL_Fps_60,                     // 60fps
    ZCL_Fps_120,                    // 120fps
    ZCL_Fps_240                     // 240fps
} ZCL_FPS;
```

Mode of extended format

```
typedef enum
{
    ZCL_Mode_0              = 0,    // Mode_0
    ZCL_Mode_1,                     // Mode_1
    ZCL_Mode_2,                     // Mode_2
    ZCL_Mode_3,                     // Mode_3
    ZCL_Mode_4,                     // Mode_4
    ZCL_Mode_5,                     // Mode_5
    ZCL_Mode_6,                     // Mode_6
    ZCL_Mode_7                      // Mode_7
} ZCL_EXTMODE;
```

Color coding ID

```
typedef enum
{
    ZCL_MONO            = 0,    // MONO 8bit
    ZCL_YUV411,                 // YUV411 8bit
    ZCL_YUV422,                 // YUV422 8bit
    ZCL_YUV444,                 // YUV444 8bit
    ZCL_RGB,                    // RGB 8bit
    ZCL_MONO16,                 // MONO16 16bit
    ZCL_RGB16,                  // RGB16 16bit
    ZCL_SMONO16,                // Signed MONO16 16bit
    ZCL_SRGB16,                 // Signed RGB16 16bit
    ZCL_RAW,                    // RAW 8bit
    ZCL_RAW16,                  // RAW16 16bit
} ZCL_COLORID;
```

Function code

```
typedef enum
{
    ZCL_BRIGHTNESS          = 0,    // Brightness
    ZCL_AE,                         // Auto Exposure
    ZCL_SHARPNESS,                  // Shaarpness
    ZCL_WHITEBALANCE,               // White Balance
    ZCL_HUE,                        // HUE
    ZCL_SATURATION,                 // Saturation
    ZCL_GAMMA,                      // Gamma
    ZCL_SHUTTER,                    // Shutter
    ZCL_GAIN,                       // Gain
    ZCL_IRIS,                       // Iris
    ZCL_FOCUS,                      // Focus
    ZCL_TEMPERATURE,                // Temperature
    ZCL_TRIGGER,                    // Trigger
    ZCL_TRIGGER_DELAY,              // Trigger Delay
    ZCL_WHITE_SHADING,              // White Shading
    ZCL_FRAMERATE,                  // FrameRate
    ZCL_ZOOM,                       // Zoom
    ZCL_PAN,                        // Pan
    ZCL_TILT,                       // Tilt
    ZCL_OPTICAL_FILTER,             // Optical Filter
    ZCL_ONE_SHOT,                   // One Shot
    ZCL_MULTI_SHOT,                 // Multi Shot
    ZCL_POWER_ONOFF,                // Power On Off
    ZCL_MEMORYCHANNEL               // Memory Channel
} ZCL_FEATUREID;
```

Trigger mode

```
typedef enum
{
    ZCL_Trigger_Mode0       = 0,    // Trigger Mode0
    ZCL_Trigger_Mode1,              // Trigger Mode1
    ZCL_Trigger_Mode2,              // Trigger Mode2
    ZCL_Trigger_Mode3,              // Trigger Mode3
    ZCL_Trigger_Mode4,              // Trigger Mode4
    ZCL_Trigger_Mode5,              // Trigger Mode5
    ZCL_Trigger_Mode14      = 14,   // Trigger Mode14
    ZCL_Trigger_Mode15              // Trigger Mode15
} ZCL_TRIGGERMODE;
```

Trigger source code

```
typedef enum
{
    ZCL_Trigger_Source0     = 0,    // Trigger Source0
    ZCL_Trigger_Source1,            // Trigger Source1
    ZCL_Trigger_Source2,            // Trigger Source2
    ZCL_Trigger_Source3,            // Trigger Source3
    ZCL_Software_Trigger    = 7     // Software Trigger
} ZCL_TRIGGERSOURCE;
```

Color conversion code

```
typedef enum
{
    ZCL_C24bit              = 0,    // 24bit Color conversion
    ZCL_C16bit,                     // 16bit Color conversion
    ZCL_C15bit,                     // 15bit Color conversion
    ZCL_CFilter,                    // No 32Bit color conversion correction of the RAW data
    ZCL_C32bit,                     // 32bit Color conversion
    ZCL_CFilterRAW8G,               // 32Bit color conversion correction of the RAW data
    ZCL_CFilterRAW16,               // No 32Bit color conversion correction of the RAW16
    data ZCL_CFilterRAW16G,         // 32Bit color conversion correction of the RAW16 data
    ZCL_CFilterRAW8 = ZCL_Cfilter   // No 32Bit color conversion correction of the RAW data
} ZCL_CONVERTMODE;
```

Row of data of RAW data

```
typedef enum
{
    ZCL_FGBRG               = 0,    // GB/RG
    ZCL_FRGGB,                      // RG/GB
    ZCL_FBGGR,                      // BG/GR
    ZCL_FGRBG,                      // GR/BG
} ZCL_CFILTERMODE;
```

Specification of selection of effective bit

```
typedef enum
{
    ZCL_SFT0            = 0,    // Subordinate position 7-0 bit
    ZCL_SFT1,                   // Subordinate position 8-1 bit
    ZCL_SFT2,                   // Subordinate position 9-2 bit
    ZCL_SFT3,                   // Subordinate position 10-3 bit
    ZCL_SFT4,                   // Subordinate position 11-4 bit
    ZCL_SFT5,                   // Subordinate position 12-5 bit
    ZCL_SFT6,                   // Subordinate position 13-6 bit
    ZCL_SFT7,                   // Subordinate position 14-7 bit
    ZCL_SFT8                    // Subordinate position 15-8 bit
} ZCL_SHIFTID;
```

Coordinates specification after color is converted

```
typedef enum
{
    ZCL_BMPmode         = 0,    // BMP file mode
    ZCL_MEMmode                 // MEMORY mode
} ZCL_STOREMODE;
```

Request code of function setting

```
typedef enum
{
    ZCL_FEATURE_OFF     = 0,    // Feature function stop
    ZCL_ONE_PUSH,               // Execution of One Push
    ZCL_AUTO,                   // Automatic setting
    ZCL_VALUE,                  // Manual value setting
    ZCL_ABSVALUE                // ABS manual value setting
} ZCL_SETREQID;
```

Status code of system call back

```
typedef enum
{
    STATUSZCL_BUSRESET          = 1,    // Bus reset generation
    STATUSZCL_POWERUP                   // Return from sleep of system
} STATUS_SYSTEMCODE;
```

3.4 Structure

Camera information

```
typedef struct
{
    UINT64          UID;                // GUID
    BYTE            VendorName[ 256 ];  // Vendor name
    BYTE            ModelName[ 256 ];   // Model name
} ZCL_CAMERAINFO, *pZCL_CAMERAINFO;
```

Camera information list

```
typedef struct
{
    DWORD           CameraCount;        // Number of connected cameras or
                                           numbers of Info arrays
    ZCL_CAMERAINFO  Info[ 0 ];          // Camera information
} ZCL_LIST, *pZCL_LIST;
```

CameraCount The number of ZCL_CAMERAINFO is specified before the request is issued. After the request is executed, the number of a number of connected cameras and effective ZCL_CAMERAINFO is set.

Camera mode setting

```
typedef struct
{
    BOOL                    StdMode_Flag;       // Standard mode flag
    union
    {
        //      Standard mode
        struct
        {
            ZCL_STDMODE             Mode;       // Mode
            ZCL_FPS                 FrameRate;  // Frame rate
        } Std;

        //      Enhancing mode
        struct
        {
            ZCL_EXTMODE             Mode;       // Mode
            ZCL_COLORID             ColorID;    // Color coding ID
            ZCL_CFILTERMODE         FilterID;   // Row of data of RAW data
        } Ext;
    } u;
} ZCL_CAMERAMODE, *pZCL_CAMERAMODE;
```

StdMode_Flag It is shown whether to use the Std definition or the Ext definition.

Enhancing mode information

```
typedef struct
{
    DWORD                   ModeCount;          // Number of enhancing modes
    struct
    {
        WORD                    MaxWidth;       // The maximum width size
        WORD                    MaxHeight;      // The maximum height size
        WORD                    C_MONO:1;       // It has the function in "1".
                                                   Hereafter, it is the same to C_RAW16.
        WORD                    C_YUV411:1;
        WORD                    C_YUV422:1;
        WORD                    C_YUV444:1;
        WORD                    C_RGB:1;
        WORD                    C_MONO16:1;
        WORD                    C_RGB16:1;
        WORD                    C_SMONO16:1;
        WORD                    C_SRGB16:1;
        WORD                    C_RAW:1;
        WORD                    C_RAW16:1;
    } Mode[ 8 ];
} ZCL_EXTMODEINFO, *pZCL_EXTMODEINFO;
```

ModeCount An effective number of Mode is shown.

Mounting function information

```
typedef struct
{
    DWORD               Version;
    ZCL_FEATUREID       FeatureID;
    BOOL                PresenceFlag;
    union
    {
        //  Function commonness definition
        struct
        {
            BYTE        Abs_Inq:1;          // Capability of control with absolute value
            BYTE        One_Push_Inq:1;     // One Push auto mode
            BYTE        ReadOut_Inq:1;      // Capability of reading the value of this feature
            BYTE        On_Off_Inq:1;       // Capability of switching this feature On and Off
            BYTE        Auto_Inq:1;         // Auto mode
            BYTE        Manual_Inq:1;       // Manual mode
            WORD        Min_Value;          // Minimum value for this feature control
            WORD        Max_Value;          // Maximum value for this feature control
            float       Abs_Min_Value;      // Absolute minimum value for this feature control
            float       Abs_Max_Value;      // Absolute maximum value for this feature control
        } Std;

        //  Trigger
        struct
        {
            BYTE        Abs_Inq:1;
            BYTE        ReadOut_Inq:1;
            BYTE        On_Off_Inq:1;
            BYTE        Polarity_Inq:1;     // Capability of changing polarity
            BYTE        Value_Read_Inq:1;   // Terminal state input function
            BYTE        Source0:1;          // Input source division
            BYTE        Source1:1;
            BYTE        Source2:1;
```

```
            BYTE        Source3:1;
            BYTE        Software:1;         // Software trigger function
            BYTE        Mode0:1;            // Mode division
            BYTE        Mode1:1;
            BYTE        Mode2:1;
            BYTE        Mode3:1;
            BYTE        Mode4:1;
            BYTE        Mode5:1;
            BYTE        Mode14:1;
            BYTE        Mode15:1;
            float       Abs_Min_Value;      // Absolute minimum value for this feature control
            float       Abs_Max_Value;      // Absolute minimum value for this feature control
        } Trigger;
        BYTE            Max_MemChannel; // The maximum memory channel number
    } u;
} ZCL_CHECKFEATURE, *pZCL_CHECKFEATURE;
```

Version The version of a present library is set.

FeatureID The function code in which it wants to check the content of the function is set.

PresenceFlag The presence of the function is set.

As for Trigger, information is set to the Trigger definition.

As for the maximum memory channel, information is set to Max_MemChannel.

Additionally, information is set to the Std definition.

Acquisition of camera parameter value

```
typedef     struct
{
    DWORD                   Version;
    ZCL_FEATUREID           FeatureID;

    union
    {
        //  Function commonness definition
        struct
        {
            BYTE            Abs:1;          // Absolute value selection
            BYTE            On_Off:1;       // Function effective/invalidity
            BYTE            Auto_M:1;       // State of automatic / Manual operation
            WORD            Value;          // Present value
            float           Abs_Value;      // Present absolute value
        } Std;

        //  WhiteBalance
        struct
        {
            BYTE            Abs:1;
            BYTE            On_Off:1;
            BYTE            Auto_M:1;
            WORD            UB_Value;       // Present value of UB
            WORD            VR_Value;       // Present value of VR
            float           Abs_Value;
        } WhiteBalance;

        //  Temperature
        struct
        {
            BYTE            Abs:1;
            BYTE            On_Off:1;
            BYTE            Auto_M:1;
```

```
    WORD        Target_Value;
    WORD        Temp_Value;
    float       Abs_Value;
} Temperature;

//  Trigger
struct
{
    BYTE        Abs:1;
    BYTE        On_Off:1;
    BYTE        Rcvd:1;
    BYTE        Polarity:1;     // Input polarity
    BYTE        Value:1;        // Input value of present
    BYTE        Source;         // Present source
    BYTE        Mode;           // Present mode
    WORD        Parameter;
    float       Abs_Value;
} Trigger;

//  TriggerDelay
struct
{
    BYTE        Abs:1;
    BYTE        On_Off:1;
    BYTE        Rcvd:1;
    WORD        Value;
    float       Abs_Value;
} TriggerDelay;

//  WhiteShading
struct
{
    BYTE        Abs:1;
    BYTE        On_Off:1;
    BYTE        Auto_M:1;
    BYTE        R_Value;
```

```
            BYTE            G_Value;
            BYTE            B_Value;
            float           Abs_Value;
        } WhiteShading;

        BOOL                Exec_Flag;
        BOOL                PowerOn_Flag;
    } u;
} ZCL_GETFEATUREVALUE, *pZCL_GETFEATUREVALUE;
```

Version The version of a present library is set.

FeatureID The function code in which it wants to acquire a present camera parameter value is set.

As for WhiteBalance, information is set to the WhiteBalance definition.

As for Temperature, information is set to the Temperature definition.

As for Trigger, information is set to the Trigger definition.

As for TriggerDelay, information is set to the TriggerDelay definition.

As for WhiteShading, information is set to the WhiteShading definition.

As for s tate of power supply, information is set to the PowerOn_Flag.

As for Multi-shot and One-shot running state, information is set to the Exec_Flag.

Additionally, a present content is set to the Std definition.

Setting of camera parameter value

```
typedef     struct
{
    DWORD                   Version;
    ZCL_SETREQID            ReqID;
    ZCL_FEATUREID           FeatureID;

    union
    {
        //      Function commonness definition
        struct
        {
            WORD            Value;
            float           Abs_Value;
        } Std;

        //      WhiteBalance
        struct
        {
            WORD            UB_Value;
            WORD            VR_Value;
            float           Abs_Value;
        } WhiteBalance;

        //      Temperature
        struct
        {
            WORD            Target;
            WORD            Temp;
            float           Abs_Value;
        } Temperature;

        //      Trigger
        struct
        {
```

```
            BYTE            Polarity:1;
            BYTE            Value:1;
            BYTE            Source;
            BYTE            Mode;
            WORD            Parameter;
            float           Abs_Value;
        } Trigger;

        //      WhiteShading
        struct
        {
            BYTE            R_Value;
            BYTE            G_Value;
            BYTE            B_Value;
            float           Abs_Value;
        } WhiteShading;
    } u;
} ZCL_SETFEATUREVALUE, *pZCL_SETFEATUREVALUE;
```

Version The version of a present library is set.

ReqID The request code is set.

FeatureID The function code in which it wants to set the camera parameter value is set.

As for WhiteBalance, information is set to the WhiteBalance definition.

As for Temperature, information is set to the Temperature definition.

As for Trigger, information is set to the Trigger definition.

As for WhiteShading, information is set to the WhiteShading definition.

Additionally, the content that the Std definition demands is set.

Image information

```
typedef         struct
{
    BOOL                StdMode_Flag;
    struct
    {
        WORD            PosX;           // Present image start position X
        WORD            PosY;           // Present image start position Y
        WORD            Width;          // Present width of image
        WORD            Height;         // Present height of image
        ZCL_COLORID     ColorID;        // Present color coding ID
        DWORD           DataLength;     // Effective data length of image data
        DWORD           Buffer;         // Necessary buffer length for image data
                                           reception
    } Image;

    struct
    {
        WORD            MaxImageX;      // The maximum image width size
        WORD            MaxImageY;      // The maximum image height size
        WORD            UnitSizeX;      // Width of cutting out unit
        WORD            UnitSizeY;      // Height of cutting out unit
        WORD            UnitPosX;       // Width of cutting out boundary
        WORD            UnitPosY;       // Height of cutting out boundary
    } Ext;
} ZCL_GETIMAGEINFO, *pZCL_GETIMAGEINFO;
```

StdMode_Flag The effectiveness of the Ext definition is set.
Information is set to the Ext definition, when the camera is set to the enhancing mode.

Specification of image information

```
typedef         struct
{
    WORD                PosX;           // Image start position X
    WORD                PosY;           // Image start position Y
    WORD                Width;          // Width of image
    WORD                Height;         // Height of image
    BOOL                MaxSize_Flag;
} ZCL_SETIMAGEINFO, *pZCL_SETIMAGEINFO;
```

MaxSize_Flag A maximum size and a minimum size of the size of the packet in the image forwarding are selected.
Maximum size TRUE / Minimum size FALSE

Register information

```
typedef         struct
{
    DWORD               Offset;
    DWORD               Size;
    PVOID               Value;
} ZCL_REGISTER, *pZCL_REGISTER;
```

Offset The Offset position from the base is specified.

Size The request size in the setting and reading is specified.

Value The area of the setting and reading is specified.

Color conversion constant

```
typedef         struct
{
    double a;
    double b;
    double c;
    double d;
} ZCL_COLORVALUE, *pZCL_COLORVALUE;
```

Color conversion execution parameter

```
typedef     struct
{
    ZCL_COLORID             ColorID;
    ZCL_CFILTERMODE         CFilter;
    ZCL_STOREMODE           StoreMode;
} ZCL_COLORMODE, *pZCL_COLORMODE;
```

ColorID	Color coding ID of the image data is set.
CFilter	When the image data is RAW data, the row and the correction, etc. of data are specified.
StoreMode	The arrangement of the data after the color is converted is specified.

The structure shown after this is for a peculiar function of Hitachi Kokusai camera.

Color information

```
typedef         struct
{
    BYTE            R;
    BYTE            G;
    BYTE            B;
    BYTE            Y;
    BYTE            C;
    BYTE            M;
} ZCL_HITACHICOLOR, *pZCL_HITACHICOLOR
```

Masking information

```
typedef         struct
{
    ZCL_HITACHICOLOR        Saturation;
    ZCL_HITACHICOLOR        Hue;
} ZCL_HMASKING, *pZCL_HMASKING;
```

4. Important Notes

4.1. Flow of Basic Operation

• Start of camera use

1. Open a camera using **ZCLOpen().**

• Preparation of communication

1. Set the camera mode and frame rate using **ZCLSetMode()**. (Can be omitted.)
2. Investigate the function installed in a camera using **ZCLCheckFeature()**. (Can be omitted.)
3. Set the required parameter using **ZCLSetFeatureValue()**. (Can be omitted.)
4. Acquire the buffer size for image data read using **ZCLGetImageInfo()**.
5. Prepare the start of communication using **ZCLIsoAlloc()**.

• Start of communication

1. Start communication using **ZCLIsoStart()**.
2. Request the acquisition of image data using **ZCLImageReq()**.
3. Confirm using **ZCLImageCompleteWait()** that the acquisition of image data is completed.
4. Repeat steps 2 and 3 as required.

• Stop of communication

1. Stop communication using **ZCLIsoStop()**.
2. Cancel the request of image data acquisition, which is not completed, using **ZCLAbortImageReqAll()**.

• Release of communication preparation

1. Release the preparation of communication using **ZCLIsoRelease()**.

• Stop of camera use

1. Close a camera (release a camera from a library) using **ZCLClose()**.

* Caution on communication

Do not shorten the interval timing between the start and stop of isochronous communication.

In case where the start and stop of isochronous communication are continuously processed, the next instructions are not accepted during image data transfer until the frame is terminated even if a camera receives stop instructions. Therefore, leave an interval of one frame until the next instructions are started after image communication stops.

[MEMO]

Technoscope Co., Ltd.

7-6-13 Kishi-cho Urawa-ku Saitama City, Saitama 336-0064

TEL. 048-822-5281 FAX. 048-822-5285