Arecont Vision

AV SDK

Interface and Usage Description

Rev 2.8.2

April 5, 2010
support@arecontvision.com

Package Contents

File Name	Description
AV2000SDK.dll	Windows shared library. Provides means of access to functions of Arecont Vision cameras from within user applications.
AV2000SDK.so	Linux shared library, verified with Redhat Linux v4.6
AV2000Types.h	Declares data types used in conjunction with AV SDK
AVLoadDll.cpp	Example of loading SDK dll
AVIntegrationExample	Demo program, MS VC++, download from Arecont Vision website
AVErrorCodes.h	Error code names
AV_SDK_v*.pdf	This file

Introduction

AV SDK is a collection of libraries and examples to connect and capture images from Arecont Vision cameras. SDK supports Windows and Linux platforms.

In the text below, a "client" is defined as the object created in the DLL that handles communication with the camera.

AV2000Types.h

Data Types

- `enum EXIF_STRING_ID {CAMERA_MAKE_STRING = 0 , CAMERA_MODEL_STRING, SOFTWARE_STRING, COPYRIGHT_STRING};`
  `CAMERA_MAKE_STRING` is used to specify the name of camera manufacturer.
  `CAMERA_MODEL_STRING` is used to specify camera model.
  `CAMERA_SOFTWARE_STRING` is used to specify the name of software that uses this SDK.
  `COPYRIGHT_STRING` is used to specify copy right owner of the image generated by this SDK.
  **These strings will be embedded into the header of every JPEG image generated by this SDK in form of EXIF (EXchangeable Image File format, a standard of Japan Electronics and Information Technology Industries Association).
  **EXIF is supported by Microsoft Windows. In Windows these information will be displayed in right-click menu\properties\summary (advanced).

- `enum IMAGE_RESOLUTION { imFULL = 1, imHALF = 2, imZOOM };`
  `imHALF` is used to request images decimated by a factor of 2 in both directions, (e.g AV2100 decimated image size is 800x600);
  `imFULL` is used to request an entire frame;
  `imZOOM` – is used to request image window (a portion of the image)

- enum `CodecID {JPEG_CODEC = 0 , H264_CODEC = 1};`
  CodecID is used in function **`GetWindowImageQEx`** to specify the way image encoder type in camera.
  `JPEG_CODEC` – corresponds to MJPEG encoder.
  `H264_CODEC` - corresponds to H.264 encoder.

-
  ```
  enum CAMERA_PARAMETER { cpBRIGHTNESS, cpSATURATION, cpSHARPNESS,
  cpBLUE, cpRED,cpILLUMINATION, cpLIGHTING, cpCAMERA_MODE,
  cpSHORT_EXPOSURES, cpQUALITY_FULL, cpQUALITY_HALF, cpQUALITY_ZOOM,
  cpDOUBLESCAN, cpROLL, cpIRIS_ENABLED, cpIRIS_SPEED, cpIRIS_GAIN,
  cpIRIS_PEROSITION_ENABLED, cpIRIS_REPOSITION_F_STOPS,
  cpIRIS_REPOSITION_F_STOPS_MIN, cpIRIS_REPOSITION_PERIOD,
  cpIRIS_REPOSITION_STABLE_PERIOD, cpDAY_NIGHT_MODE,
  cpDAY_NIGHT_TRIGGER_NIGHT, cpDAY_NIGHT_TRIGGER_DAY, cpEXPOSURE_MODE,
  cpEXPOSURE_WINDOW_LEFT, cpEXPOSURE_WINDOW_TOP, cpEXPOSURE_WINDOW_WIDTH,
  cpEXPOSURE_WINDOW_HEIGHT,cpSENSOR_LEFT, cpSENSOR_TOP, cpSENSOR_WIDTH,
  cpSENSOR_HEIGHT,cpSENSOR_BLACK_WHITE_LEFT,
  cpSENSOR_BLACK_WHITE_TOP,cpSENSOR_BLACK_WHITE_WIDTH,
  cpSENSOR_BLACK_WHITE_HEIGHT,cpREQUEST_LEFT, cpREQUEST_TOP,
  cpREQUEST_WIDTH, cpREQUEST_HEIGHT,cpQUALITY, cpRESOLUTION,
  cpPER_CENT_IMAGE_RECTANGLE, cpREQUESTED_BLOCK_SIZE, cpMD_ENABLED,
  cpMD_MODE, cpMD_LEVEL_THRESH, cpMD_TOTAL_ZONES, cpMD_ZONE_SIZE,
  cpMD_EXPLOSURE_SENSITIVITY, cpMD_MATRIX, cpMD_DETAIL, cpH264,
  cpBIT_RATE, cpGAMMA, cpBIN_DAY, cpBIN_NIGHT, cpBIN_1080P }
  ```

Parameter	Description	Values	Default[1]	Stored
cpBRIGHTNESS	Sets target image brightness	-99… 99	5	Camera
cpSATURATION	Color saturation, % of normal saturation	0, 25, 37, 50, 75, 100, 125	100	Camera
cpSHARPNESS	Image sharpening	0..4	2	Camera
cpBLUE	Blue AWB target	-10..10	0	Camera
cpRED	Red AWB target	-10..10	0	Camera
cpILLUMINATION	0 – auto 1 – indoor illumination 2 – outdoor illumination 3 – mixed illumination	0..3	0	Camera
cpLIGHTING	Power supply frequency, 50 or 60Hz	50 or 60	50	Camera
cpCAMERA_MODE	Determines low-light camera operation 0 – maintain higher frame rate 1 – higher image quality 2 – balanced 3 – custom mode (read only value, to set use SetCustomMode) 4 – HIGH_SPEED mode - fixed shutter widths (set by cpSHORT_EXPOSURES) 5 – MoonLight™ mode – long exposures with proprietary noise cancellation	0..5	0	Camera
cpSHORT_EXPOSURES	Sets fixed shutter width in ms; Only valid with cpCAMERA_MODE = 4	1…80	5	Camera
cpQUALITY_FULL	JPEG compression quality in imFULL mode. Higher value corresponds to higher quality	1..21	15	DLL[2]

[1] Some defaults are stored in the camera's non-volatile memory and may become different from the ones shown in the table.
[2] If camera-stored jpeg quality setting is required, use cpQUALITY with GetDefaultImage

Parameter	Description	Values	Default	Stored
cpQUALITY_HALF	JPEG compression quality in imHALF	1..21	15	DLL[2]
cpQUALITY_ZOOM	Compression quality in imZOOM mode	1..21	15	DLL[2]
cpDOUBLESCAN	0 – requesting an image from a camera will obtain a newly captured image. The camera will delay its response until the new image is captured 1 – the camera will output last buffered image if the new image is not yet available. Use this mode to request multiple sub-windows from the same image	0, 1	0	DLL
cpREQUESTED_BLOCK_SIZE	Allows the user to specify whether camera should transmit images via single part TFTP packets (1450 bytes) or multi-part TFTP packets (2904 bytes); Valid for firmware versions 52109 or above;	1450, 2904	2904	Camera
cpROLL	180 – rotate image by 180 degrees. Use this mode if the camera is mounted upside-down	0, 180	0	Camera
cpEXPOSURE_MODE	Controls auto backlight compensation and automatic vs. user-defined exposure measurement window; 0 – automatic mode without backlight compensation 1 – automatic mode with backlight compensation 2 – user-controlled exposure window without backlight compensation	0,1,2	1	Camera
cpEXPOSURE_WINDOW_LEFT	Left coordinate of user-defined auto-exposure measurement window. Valid then cpEXPOSURE_MODE = 2	Larger than cpSENSOR_ LEFT	32	Camera
cpEXPOSURE_WINDOW_TOP	Top coordinate of user-defined auto-exposure measurement window. Valid then cpEXPOSURE_MODE = 2	Larger than cpSENSOR_ TOP	32	Camera
cpEXPOSURE_WINDOW_WIDTH	Width of user-defined auto-exposure measurement window. Valid then cpEXPOSURE_MODE = 2	Smaller than cpSENSOR_ WIDTH – cpSENSOR_ LEFT	cpSENSOR _ WIDTH - 64	Camera

Parameter	Description	Values	Default	Stored
cpEXPOSURE_WINDOW_HEIGHT	Height of user-defined auto-exposure measurement window. Valid then cpEXPOSURE_MODE = 2	Smaller than cpSENSOR_ HEIGHT – cpSENSOR_ TOP	cpSENSOR _ HEIGHT - 64	Camera
cpSENSOR_LEFT	Left coordinate of sensor window	Up to sensor width	0	Camera
cpSENSOR_TOP	Top coordinate of sensor window	Up to sensor height	0	Camera
cpSENSOR_WIDTH	Width of sensor window	320 … up to sensor width	Model-dependent See Table 1	Camera
cpSENSOR_HEIGHT	Height of sensor window	240 … up to sensor height	Model-dependent See Table 1	Camera
cpWIDTH	Image width returned by functions GetImage and GetImage2	Up to `cpSENSOR_WIDTH`	AV1300 – 1280 AV2100 - 1600 AV3100 – 1920 AV5100 – 2560 AV3130 - 1920	SDK
cpHEIGHT	Image height returned by functions GetImage and GetImage2	Up to `cpSENSOR_HEIGHT`	AV1300 – 1024 AV2100 - 1200 AV3100 – 1200 AV5100 – 1600 AV3130 - 1200	SDK
cpREQUEST_LEFT	Default image left coordinate relative to sensor window for GetDefaultImage().	Up to sensor width	0	Camera
cpREQUEST_TOP	Default image top coordinate relative to sensor window for GetDefaultImage().	Up to sensor height	0	Camera
cpREQUEST_WIDTH	Default image width for GetDefaultImage().	320 … up to sensor width	Default cpSENSOR _WIDTH	Camera
cpREQUEST_HEIGHT	Default image height for GetDefaultImage().	240 … up to sensor height	Default cpSENSOR _HEIGHT	Camera
cpQUALITY	Default image quality for GetDefaultImage()	1…20	15	Camera
cpRESOLUTION	Default image resolution for GetDefaultImage()	Full, half (0,1)	1	Camera
cpH264	Use it to find out if H.264 encoder enable or not.	0 – disable 0> - enable	Depends on the model	SDK
cpAUTO_EXPOSITION	0 – disable (freeze) auto-exposure 1 – enable auto-exposure	0,1	Always 1 after reset	n/a
cpGAMMA	Round(100/gamma)	40..100 gamma2.5~1.0	60 gamma=1.67	Camera

Parameter	Description	Values	Default	Stored
AV10005 Dual-mode Camera Specific Parameters				
cpBIN_DAY	Binning in Day mode	0,1	0	Camera
cpBIN_NIGHT	Binning in Night mode	0,1	1	Camera
cpBIN_1080P	1080p mode	0,1	0	Camera
On-Camera Motion-Detection Parameters[3]				
cpMD_ENABLED	Enables detection '0' – disabled '1'– enabled	0,1	0	Camera
cpMD_MODE	Motion detection mode '1' – camera returns the image of zero length in the absence of motion; '0' – camera always returns requested image	0,1	0	Camera
cpMD_LEVEL_THRESH	Motion threshold (sensitivity to change)	2..255	3	Camera
cpMD_TOTAL_ZONES	Number of independent motion detection zones; Currently must be 64. There are 8 rows of zones, 8 zones per row	64	64	Camera
cpMD_ZONE_SIZE	Size of motion detection zones measured in number of 32x32 pixel blocks in each zone; All zones are squares of the same size from 7x7 to 15x15	7..15 (8…15 for AV3100)	8	Camera
cpMD_EXPLOSURE_SENSITIVITY	Sensitivity to overall brightness changes. If more than this number of zones have motion it is assumed that the change is due to illumination change and detected motion is ignored;	0..64	40 for AV3100 30 for AV2100 20 for AV1300	Camera
cpMD_MATRIX	Privacy matrix. 8-byte array, where each byte corresponds to one row of motion detection zones. Each bit in a byte enables motion detection in a corresponding zone, if set to '1'. Leftmost zone is		All "1'	Camera

[3] By the time of this manual, on-camera motion detection is available on all AV cameras including AV8360, AV8180, AV8365 and AV8185. Some older models (for example old AV5100) require firmware/hardware upgrade to support on-camera motion detection.

Parameter	Description	Values	Default	Stored
	controlled by MSB, rightmost zone by LSB. Also see Motion Detection section. [4]			
cpMD_DETAIL	Allows to control the size of moving objects to be detected. The value is the number of 32x32 sub-zones within each zone that should contain motion for the entire zone to be considered to contain motion.	1 through square of zone size	1	Camera
Auto-Iris Parameters[5]				
cpIRIS_ENABLED	Enable auto-iris '0' – disabled Non-zero value – enabled	0,non-zero	0	Camera
cpIRIS_SPEED	Iris servo-loop gain. Always keep at default setting.	1..255	64	Camera
cpIRIS_GAIN	Maximum gain to be used for exposure when auto-iris is enabled. Gain = x/8	8..255	20	Camera
cpIRIS_REPOSITION_ F_STOPS	Number of f-stops to close, in half-steps: N[f-stops] = (x+1)/2.	1..15	5[6]	Camera
cpIRIS_REPOSITION_ F_STOPS_MIN	Minimum number of f-stops to close, in half-steps: N[f-stops] = (x+1)/2.	1..15	1	Camera
cpIRIS_REPOSITION_ PERIOD	Approximate time between repositioning. T[minutes] = x/20	0..900	600 (2M) 500 (3M)	Camera
cpIRIS_REPOSITION_ STABLE_PERIOD	Approximate time to evaluate scene before closing the iris. T[minutes] = x/20	0..900	1	Camera
Single Sensor and Panoramic DayNight model Specific Parameters				
cpDAY_NIGHT_MODE	0 = automatic 1 = day 2 = night	0,1,2	Always 0 after reset	Camera
cpDAY_NIGHT_ TRIGGER_NIGHT	Gain threshold for switching from day mode to night mode	64..512	256	Camera
cpDAY_NIGHT TRIGGER_DAY	When brightness increase to 1+ 2^(-x) times the	1..4	2	Camera

[4] For detailed description of cpMD_MATRIX usage reference Motion Detection section of this manual.
[5] Auto-iris parameters do not apply to AV3130
[6] This value depends on the model of the auto-iris lens

Parameter	Description	Values	Default	Stored
	darkest point, return to day mode.			
AV3130/AV3135 Day-Night Specific Parameters				
cpDAY_NIGHT_MODE	AV3130/3135 mode: - 0 = automatic - 1 = day - 2 = night	0,1,2	Always 0 after reset	n/a
cpDAY_NIGHT_ TRIGGER_NIGHT	AV3130/3135 exposure threshold to enter the night mode Fstops = 2^(x / 2.0)	0..18	3	Camera
cpDAY_NIGHT TRIGGER_DAY	AV3130/3135 hysteresis to return to day mode Fstops = 2^(x / 2.0)	0..6	2	Camera
cpSENSOR_BLACK_ WHITE_LEFT	Left coordinate of monochrome sensor window	Up to 1280	0	Camera
cpSENSOR_BLACK_ WHITE_TOP	Top coordinate of monochrome sensor window	Up to 1024	0	Camera
cpSENSOR_BLACK_ WHITE_WIDTH	Width of monochrome sensor window	Up to 1280	1280	Camera
cpSENSOR_BLACK_ WHITE_HEIGHT	Height of monochrome sensor window	Up to 1024	800	Camera
cpPER_CENT_ IMAGE_RECTANGLE	If set, GetWindowImage parameters are specified as percentage of sensor window	0,1	0	Camera
Panoramic Specific Parameters (AV8360/8180/8365/8185)				
cpMS_NUMBER_OF_SENSOR	Sensor (channel) selected for setting its parameters	1,2,3,4	0	SDK
cpMS_CHANNEL_ENABLE	A 4-bit mask to select a combination of sensors (channels) that are enabled for sending images	0x0001-0x000F	0x000F	Camera
cpMS_FULL_RES_ENABLE	A 4-bit mask to select a combination of sensors (channels) that send full resolution images; all other enabled channels send decimated images	0x0000-0x000F	0x0000	Camera
cpMS_ZOOM_WIN_ENABLE	A 4-bit mask to select a combination of sensors (channels) that send zoom window images	0x0000-0x000F	0x0000	Camera
cpMS_ONE_SHOT_ENABLE	A 4-bit mask to select a combination of sensors (channels) that send a	0x0000-0x000F	0x0000	Camera

Parameter	Description	Values	Default	Stored
	single full resolution image only once. To repeat, set the parameter again.			
cpMS_IS_ZOOMED	Indicates whether the image received is a zoom window.	0,1	0	n/a
cpMS_QUAD_MODE	Enables a special camera mode in which each channel sends a full resolution image followed by four decimated images from all channels.	0,1	0	Camera
cpBIT_RATE	Desired bit rate for H.264 streams.	0..65000	0	Camera

In the table above some defaults are stored in the camera non-volatile memory. Other parameters apply to DLL and are lost when the program terminates and DLL is unloaded. The application programmer is responsible for saving these settings and setting them again the next time DLL is loaded.

NOTE: Parameters described as stored on camera are updated in the camera non-volatile memory only AFTER THE CALL TO Permanently(). Mere writing a new value using SetAV200Parameter does not modify non-volatile memory. Permanently() function should NOT be called more than 20,000 times in a camera lifetime.

```
struct AV2000Addr{
   unsigned char ip[4],
                 mac[6];
}
```

This structure contains camera Ethernet MAC and associated IP.

- `enum IRIS_STATUS { irUNKNOWN = 0, irMANUAL, irDISABLED, irIDLE, irEVALUATING, irEVALUATING_TOO_DARK, irCLOSING, irCLOSED, irOPENING };`

  `irUNKNOWN` – iris status cannot be determined
  `irMANUAL` – iris operates in manual mode
  `irDISABLED` – iris is disabled or iris cable is unplugged
  `irIDLE` – iris is idle
  `irEVALUATING` – camera is evaluating the scene before closing the iris
  `irEVALUATING_TOO_DARK` – same as irEVALUATING, but previous evaluation concluded that the scene is too dark to allow the iris to close
  `irCLOSING` – the iris is closing down
  `irCLOSED` – the iris is closed down
  `irOPENING` – the iris is opening

Errors

```
struct ClientError{
int code; char description[256];
};
```

The structure is used by AV client to communicate encountered errors, where

`code` – is the error code. Currently supported values are:

<0 – fatal error, caused by DLL code

=0 – error code is undefined.

The application should terminate when receiving an error code less or equal to 0, since DLL behavior may become unpredictable

>0 – recognized internal DLL error, typically caused by incorrect arguments passed to DLL functions or a network communication error

`description` – ASCII error description.

Errors returned by the ClientError structure can originate either from OS system or from DLL code. System errors have a code and description defined by the operating system. Windows error codes can be found in MSDN documentation. <error.h> defines Linux errors.
DLL errors are issued by DLL's own code and are defined in AVErrorCodes.h.
The kind of code returned in "code" field is determined by the value of bit 0 of description[256]. A '1' indicates DLL code error, a '0' indicates OS system error.

Error Handling Example

It is recommended that the user verifies the success of each call to SDK functions that have client number as an input parameter by subsequent call to GetLastClientError(number_of_client);

```
// declarations
// initialization
// create client
// main loop
// ...
ClientError* err = pGetLastClientError(number_of_client);
if(err->code > 0){
    if(err->description[255] & 1){
        // see AVErrorCodes.h for identification
        // your code
    }
    else{
        // see MSDN into windows or error.h into linux
        // your code
    }
}
else{
    // fatal error - terminate, restart program
    // your code
}
```

Error Codes defined in AVErrorCodes.h

```
ZERO_POINTER                    = 1,
NO_ALLOCATED_MEMORY             = 2,
BAD_SOCK_ADDRESS               = 3,
TIMEOUT_ON_SOCKET              = 4,
READ_STRING_FROM_SOCKET        = 5,
TFTP_PROTOCOL                  = 6,
CAMERA_PARMATER_OUT_OF_RANGE   = 7,
UNKNOWN_CAMERA                 = 8,
PARAMETER_NOT_SUPPORTED        = 9,
VALUE_OF_PARAMETER_UNKNOWN     = 10,
UNKNOWN_ERROR                  = 11,
CLIENT_ALREADY_EXISTS          = 12,
CLIENT_OUT_OF_RANGE            = 13,
CLIENT_DOES_NOT_EXIST          = 14,
TFTP_MISSING_PACKETS           = 15;
```

Note that error `TFTP_MISSING_PACKETS` indicates that image has been received, but some packets might be missing, resulting in lost image segments.

AV2000SDK.dll and AV2000SDK.so

List of Functions

The shared libraries contain the following functions:

```
#define export_dll_function __declspec(dllexport)

// Client functions
extern "C" export_dll_function int CreateClient(int number);
extern "C" export_dll_function void DestroyClient(int number);
extern "C" export_dll_function unsigned MaxClients();
extern "C" export_dll_function const ClientError* const GetLastClientError(int number);
extern "C" export_dll_function int SetClientIp(int number, const char* ip);
extern "C" export_dll_function const char* GetClientIp(int number);
extern "C" export_dll_function void SetClientPort(int number, unsigned port);
extern "C" export_dll_function void SetClientTimeout(int number, unsigned value);
extern "C" export_dll_function unsigned GetClientTimeout(int number);
extern "C" export_dll_function unsigned GetClientPort(int number);
extern "C" export_dll_function int GetImage(int number, char** pbuffer, unsigned long* psize,
IMAGE_RESOLUTION resolution, float zoom, int dx, int dy);
extern "C" export_dll_function int GetImage2(int number, char** pbuffer, unsigned long* psize, unsigned
long* pcapacity, IMAGE_RESOLUTION resolution, float zoom, int dx, int dy);
extern "C" export_dll_function int GetWindowImage(int number, char** pdata, unsigned long* size, unsigned
long* capacity,  IMAGE_RESOLUTION, int left, int top, int width, int height);
extern "C" int export_dll_function STDCALL GetWindowImageQ(int number, char** pdata, unsigned long* size,
unsigned long* capacity,  IMAGE_RESOLUTION resolution, long IsPercent, long Aquality, int left, int top,
int width, int height);

extern "C" int export_dll_function STDCALL GetWindowImageQEx(int number, char** pdata, unsigned long* size,
unsigned long* capacity,  IMAGE_RESOLUTION, long IsPercent, long Aquality, int left, int top, int width, int
height, int codec, int streamId, int* Iframe);


extern "C" export_dll_function int GetDefaultImage(int number, char** pdata, unsigned long* size, unsigned
long* capacity);
```

```
extern "C" int export_dll_function STDCALL GetDefaultImageEx(int number, char** pdata, unsigned long* size,
unsigned long* capacity, int codec, int streamId, int* Ifarme);

extern "C" export_dll_function int SetAV2000Register(int, unsigned char registr, unsigned char* value);
extern "C" export_dll_function int GetAV2000Register(int, unsigned char registr, unsigned char* value);
extern "C" export_dll_function int SetAV2000Parameter(int number, CAMERA_PARAMETER parameter, long* pvalue);
extern "C" export_dll_function int GetAV2000Parameter(int number, CAMERA_PARAMETER parameter, long* pvalue);
extern "C" export_dll_function int Permanently(int number);
extern "C" export_dll_function int FactoryDefault(int number);
extern "C" export_dll_function int UpdateVersion(int number);
extern "C" export_dll_function unsigned long Model(int);
extern "C" export_dll_function unsigned long Version(int number);
extern "C" export_dll_function unsigned long Revision(int number);
extern "C" export_dll_function unsigned long DayNight(int number);
extern "C" export_dll_function unsigned long Dome(int number);
extern "C" export_dll_function unsigned long Compact(int number);
extern "C" export_dll_function int SetCustomMode(int number, long* pknee_point, long* pmax_analog_gain,
long* pmax_knee_gain, long* pmax_exposure_time, long* pmax_digital_gain);
extern "C" export_dll_function int GetCustomMode(int number, long* pknee_point, long* pmax_analog_gain,
long* pmax_knee_gain, long* pmax_exposure_time, long* pmax_digital_gain);
extern "C" export_dll_function int IsBlackWhite(int number, int* pblack_white);
extern “C” export_dll_function int GetMac(AV2000Addr*);
extern "C" export_dll_function void SetClientBuffer(int number, char* buffer, unsigned long size);

typedef void (*PTR_Allocate)(char**, unsigned long*);
typedef void (*PTR_Deallocate)(char*);
typedef void (*PTR_Reinited)(int number);

// Memory allocation support function
extern "C" export_dll_function void SetAllocateFunction(PTR_Allocate);
extern "C" export_dll_function void SetDeallocateFunction(PTR_Deallocate);
extern "C" export_dll_function void SetReinitedFunction(PTR_Reinited);

// Camera detection and IP configuration functions
extern "C" export_dll_function int FindCameras(unsigned* number, unsigned timeout);
extern "C" export_dll_function int GetCameras(AV2000Addr*, unsigned* psize);
extern "C" export_dll_function int SetCameraIp(AV2000Addr*);
extern "C" export_dll_function const ClientError* const GetLastSetError();
extern “C” export_dll_function int CheckCamera(const char* ip);

// Auto-Iris functions
extern "C" export_dll_function int IrisPresent(int);
extern “C” export_dll_function int GetIrisStatus(int);

//Motion Detection Information
extern “C” export_dll_function const unsigned char* GetMotionArray(int number)

//Auxiliary IO Functions
extern "C" export_dll_function int SetAuxIO(int number, int value);
extern "C" export_dll_function int GetAuxIO(int number);
extern "C" export_dll_function int GetAuxOutStatus(int number);

//Single Capture Mode Functions
extern "C" export_dll_function int TriggerSingleCapture(int number);
extern "C" export_dll_function int GetSingleCapture(int number);
extern "C" export_dll_function int SetSingleCapture(int number, int value);
extern "C" export_dll_function int SetCalibrateFlash(int number);
extern "C" export_dll_function int GetCalibrationNumber(int number);

//EXIF CopyRight Strings Functions
extern "C"  export_dll_function int InitializeCopyRightStrings(int, EXIF_STRING_ID, unsigned char*);
```

Camera Communication Functions

For all functions "number" is the input argument specifying a client number. If a function returns an int, a positive value indicates a successful completion, value of 0 indicates an error during execution. The error code is accessed via GetLastSetError() returning a ClientError pointer.

- `void` **`CreateClient`**`(int number)`

Creates an instance of AV2000 client. Returns 0 if success. Returns a positive value if recoverable error, see AVErrorCodes.h or negative if fatal error. The client number must be greater than 0 and less than MaxClients()-1.

- `void` **`DestroyClient`**`(int number)`

Destroys the client and frees its memory, provided that the client exists. Otherwise no operation is performed.

- `unsigned` **`MaxClients`**`()`

Returns maximum number of clients that can be created.

- `const ClientError* const` **`GetLastClientError`**`(int number)`

Reports last error detected by the client.

- `void` **`SetClientIp`**`(int number, const char* ip)`

Sets IP address of the client. The address must match the IP of a camera, where ip is specified as e.g. "192.168.254.10".

- `char*` **`GetClientIp`**`(int number)`

Returns the client's IP address.

- `unsigned` **`GetClientPort`**`(int mumber)`

Returns the client's port. Currently always returns 69, the number of TFTP port.

- `int` **`GetImage`**`(int number, unsigned char** buffer, unsigned long* psize, IMAGE_RESOLUTION resolution, float zoom, int dx, int dy)`

Returns camera image in jpeg format. Note that for zoom = 1 the returned image size is determined by the values of cpHEIGHT and cpWIDTH with default size being smaller than maximum possible value in the case of AV3100, AV5100 and AV3130.

  - o pbuffer –pointer to the memory buffer containing the image (input/output parameter);
  - o psize – pointer to the size of received image;
  - o resolution – imFULL, imHALF, imZOOM as described above;
  - o zoom – determines the size of the window requested from the camera. Valid values are from 1.0 to 0.05. If zoom=1.0 and resolution=imFULL then the full image is requested. For zoom < 1.0, windows of smaller sizes are returned. For example, resolution=imFULL and zoom=0.25 returns the window of width 1600*0.25=400 and height 1200*0.25=300. The received image may not exactly match the requested due to camera restrictions.
  - o dx, dy – define the position of the window requested from the camera (for zoom < 1.0). The coordinates of the received window are given by:

    Left = 1600*(1.0 – zoom)*(50 – dx)/100; Right = Left + 1600*zoom;

    Top = 1200*(1.0 – zoom)*(50 – dy)/100; Bottom=Top + 1200*zoom;

    Valid dx,dy values are from -50 to +50. (-50,-50) corresponds to lower left corner.

    Due to certain requirements imposed by the camera on the window alignment, the resulting window size and position may differ slightly from the requested values.

- `int` **`GetImage2`**`(int number, unsigned char** buffer, unsigned long* psize, unsigned long* pcapacity, IMAGE_RESOLUTION resolution, float zoom, int dx, int dy)`

Same as GetImage but has an added parameter pcapacity that returns the size of the buffer used for acquiribg the image. Note that for zoom = 1 the returned image size is determined by the values of cpHEIGHT and cpWIDTH with default size being smaller than maximum possible value in the case of AV3100, AV5100 and AV3130.

- o pcapacity is the full size of the image buffer while psize is the size of the image as above, wherein psize is always less or equal to pcapacity;

• `extern "C" export_dll_function int` **`GetWindowImage`**`(int number, char** pdata, unsigned long* size, unsigned long* capacity, IMAGE_RESOLUTION resolution, int left, int top, int width, int height),`
Returns a rectangular sub-window of camera image;

- o pdata, size, pcapacity and resolution have the same meaning as in GetImage();
- o left, top, width and height define the size of the image sub-window being requested, where left and top are in coordinates relative to cpSENSOR_LEFT and cpSENSOR_HEIGHT. Note: parameters left and width should be integer multiples of 32, while parameter top and height should be integer multiple of 16. Further, parameter width can not be less than 320 while parameter height can not be less than 240; For example, the following code results in transmitted image window that has the width and height of 1024 and has its left boundary at the sensor column 32 and top boundary at the senor row 64:

```
SetAV2000Parameter(cpSENSOR_LEFT, 32);
SetAV2000Parameter(cpSENSOR_TOP, 64);
SetAV2000Parameter(cpSENSOR_WIDTH, 1024);
SetAV2000Parameter(cpSENSOR_HEIGHT, 1024);
char *data = new char[1000000];
unsigned long size = 1000000,capacity;
int left = 0, top = 0,width = 1024,height = 1024;
IMAGE_RESOLUTION res = imFULL;
GetWindowImage(1, &data, &size, &capacity, res, left, top, width, height)
```

• `extern "C" export_dll_function int` **`GetWindowImageQ`**`(int number, char** pdata, unsigned long* size, unsigned long* capacity, IMAGE_RESOLUTION resolution, long IsDoubleScan, long Aquality, int left, int top, int width, int height),`
Returns a rectangular sub-window of camera image;

- o pdata, size, pcapacity and resolution have the same meaning as in GetWindowImage();
- o left, top, width and height have the same meaning as in GetWindowImage();
- o IsDoubleScan, 0 or 1.
  * if this parameter is set to 1, this function will return an image that is AVAILABLE IMMEDIATELY. The same image could have been sent out multiple times, or it could be a new image just arrived in the buffer.
  * if this parameter is set to 0, this function will return a NEW image. The image could be immediately available if the current buffer is never read before. Otherwise this function will wait until a new image is ready to be sent out;
- o Aquality, quailty of requested image, 1~21, recommended value: 15.

• `extern "C" int export_dll_function STDCALL` **`GetWindowImageQEx`**`(int number, char** pdata, unsigned long* size, unsigned long* capacity, IMAGE_RESOLUTION, long IsPercent, long Aquality, int left, int top, int width, int height, int codec, int streamId, int* Iframe, int bitrate, int intra_period);`

Returns a rectangular sub-window of camera image. In comparison with GetWindowImageQ it has five additional parameters.

- o codec – is an input parameter which specifies codec type. 0 – MJPEG; 1 – H.264. Parameter value of 1 applies to camera models that support H.264 and end with "05", for example "AV2105".

- o streamId – is an input parameter which specifies stream identifier. MPEG codecs including H.264 codecs are context dependent (decoding of the current frame depends on the previous frame(s)). The stream is a sequence of frames of the same size which can be decoded sequentially by one instance of an MPEG decoder. Accordingly, streamId is a means to distinguish different streams from each other. Use unique streamId for each stream with a unique image size and/or frame rate. Value range of streamId is 0 through 65535. Camera supports 8 simultaneous streams. Each individual client must have a unique streamId. Parameter streamId will be ignored in case of JPEG codec.

- Iframe – is an input & output parameter. If Iframe is set to 1 the camera will return an Intra frame with a corresponding SPS and PPS as an IDR slice – so that the stream is decodable from this point. When opening a new stream (for example when changing the image size and/or frame rate) the Iframe parameter will be set automatically set to 1 regardless of the input value of Iframe. To minimize the stream size use Iframe = 1 as rarely as possible. The number of P-frames is set using SetAV2000Register with register=21, value=number of P-frames. The camera will return an Intra frame even if Iframe is set to 0 when the on-camera counter of P-frames fills up. To find out whether an Intra frame was received, check the value of Iframe after the function call. Parameter Iframe is ignored in case of JPEG codec type.

- bitrate – is an input parameter. This parameter is used to set up bitrate of the stream in kilobits per second. This parameter is only available for H.264 streams (ignored in case of JPEG stream). **If this parameter is non-zero then Aquality parameter will be ignored and camera will adjust quantization parameters toward desired bit rate.**

- intra_period – is an input parameter. intra_period should be equal to intra coded frame period. This parameter is used in rate control.This parameter is only available for H.264 stream(it is ignored in case of JPEG stream). If you are not use your own counter of inter coded frames and do not set up Iframe to 1(register 3:21 is used) then you should use zero value. Let it be I – intra coded frame; P – inter coded frame. I P P P I - intra_period must be 4; I I I I - intra_period must be 1.

• `extern "C"  int` **`GetDefaultImage`**`(int number, char** pdata, unsigned long* size, unsigned long* capacity)`

Returns a rectangular sub-window of full camera image, with the size, compression quality and resolution of returned of sub-window determined by the values of parameters cpRESOLUTION, cpQUALITY, cpREQUEST_LEFT, cpREQUEST_TOP, cpREQUEST_WIDTH, cpREQUEST_HEIGHT;

- o pdata, size, pcapacity have the same meaning as in GetImage();
- o This function operates with cameras AV1300, AV2100 and AV3100, revision 5 and above, firmware version 52108 and above.
- o For example, the following code results in transmitted image window that has the width and height of 1024 and has its left boundary at the full image column 32 and top boundary at the full image row 64:

```
SetAV2000Parameter(cpREQUEST_LEFT, 32);
SetAV2000Parameter(cpREQUEST_TOP, 64);
SetAV2000Parameter(cpREQUEST_WIDTH, 1024);
SetAV2000Parameter(cpREQUEST_HEIGHT, 1024);
char *data = new char[1000000];
unsigned long size = 1000000,capacity;
int left = 0, top = 0,width = 1024,height = 1024;
GetDefaultImage(1, &data, &size);
```

- `int` **`SetAV2000Register`**`(int number, unsigned char register, unsigned char* value)`

Set AV2000 register to the value "value";
  - register – camera register number – valid values are 0..255,
  - value – register value, 2-element byte array, first element contains MSB, second element contains LSB.

- `int` **`GetAV2000Register`**`(int number, unsigned char register, unsigned char* value)`

Read AV2000 register "register".

- `int` **`SetAV2000Parameter`**`(int number, CAMERA_PARAMETER parameter, long* pvalue)`

Set a camera parameter described in enum CAMERA_PARAMETER.
  - pvalue – pointer to the value of the camera parameter (input)

- `int` **`GetAV2000Parameter`**`(int number, CAMERA_PARAMETER parameter, long* pvalue)`

Returns the current value of the specified camera parameter.

- `int` **`Permanently`**`(int number)`

Save current values of camera registers into camera non-volatile memory. The camera will use these settings after a power-off.

NOTE: The use of this function is the ONLY way to store any parameters in the non-volatile memory. This function takes a few seconds to execute and it should not be called more than 20,000 times over camera lifetime, to prevent permanent damage to non-volatile memory.

- `int` **`FactoryDefault`**`(int number)`

Set camera and library parameters to their default values. Note that default values are loaded in the camera but are not saved permanently. To restore factory settings of the camera invoke function Permanently after invoking FactoryDefault.

- `int` **`SetCustomMode`**`(int number, long* pknee_point, long* pmax_analog_gain, long* pmax_knee_gain, long* pmax_exposure_time, long* pmax_digital_gain)`

Set custom values for parameters of AE and AGC algorithm. Note that the use of this function is optional and for most users pre-defined values of cpCAMERA_MODE will produce good results.
  - `pknee_point` – pointer to the value that specifies preferred exposure time. The camera will not increase exposure time above knee_point until gain reaches the value pointed to by `pmax_knee_gain;` This parameter is specified in terms of the multiple of 10 ms increments for 50Hz setting, or 8.33 ms for 60Hz setting. The valid range is from 1 to 100.
  - `pmax_analog_gain` – pointer to the value of maximum analog gain that will be used by the AGC algorithm; The valid range of this parameter is from 1 to 10;
  - `pmax_knee_gain` – pointer to the value that specifies maximum gain (analog*digital) that will be used by the AGC algorithm while the exposure time is equal to *pknee_point; This parameter is an integer value in the range from 2 `to (*pmax_analog_gain)*(*pmax_digital_gain/32)`
  - `pmax_exposure_time` – pointer to the value of maximum exposure time that will be used by the AGC algorithm. This parameter is an integer value no greater than 100.
  - `pmax_digital_gain` – pointer to the value that specifies maximum digital gain that will be used by the AGC algorithm. This parameter is an integer value in the range from 32 to 127, where 32 corresponds to gain of 1x, 64 corresponds to gain 2x, etc.

Example:
Setting cpCAMERA_MODE = 1 is equivalent to using SetCustomMode() with *pknee_point = 4, *pmax_analog_gain = 10, *pmax_knee_gain = 5, *pmax_exposure_time = 20 and *pmax_digital_gain = 96;
In this example, as the illumination decreases, camera will first increase exposure time to 40 ms, while maintaining gains close to unity, after that the camera will raise gain (in this case analog only) to 5x, after that the camera will increase exposure time to 200 ms, after that the camera will increase the analog gain to 10x and finally, the camera will increase digital gain to 3x.

• `void int` **`GetCustomMode`**`(int number, long* pknee_point, long* pmax_analog_gain, long* pmax_knee_gain, long* pmax_exposure_time, long* pmax_digital_gain)`
Returns current values of `*pknee_point, *pmax_analog_gain, *pmax_knee_gain, *pmax_exposure_time and *pmax_digital_gain;`

• `extern "C" export_dll_function int` **`IsBlackWhite`**`(int number, int* pblack_white)`
Returns color information about the last image obtained from AV3130 camera`;`

If *pblack_white is 0 then the last image was color image, otherwise the last received image was monochrome image. The monochrome image output by AV3130 appears as color image, as it has 3 color components, equal to each other. This function is meant to assist the user in switching decompressor parameters, as monochrome decompression is less cpu-intensive.

• `void` **`SetClientBuffer`**`(int number, char* buffer, unsigned long size)`
Sets image buffer used by the client to output the image.
- o buffer – pointer to the image buffer
- o size – size of the image buffer pointed to by "buffer"

• `typedef void (*`**`PTR_Reinited`**`)(int number)`
Pointer to a function invoked during camera initialization or when client is connected or when the camera was powered off and on. The function is supplied by the application programmer and is called by the SDK DLL. "number" is the client number that identified camera power-on/off condition.

• `void` **`SetReinitedFunction`**`(PTR_Reinited)`
Specifies the function that the client will use in the event of abrupt camera re-initialization (e.g. due to power interruption). This function has to be called prior to using the client.

• `const ClientError* const` **`GetLastSetError`**`()`
Obtain the last error that occurred after calling a camera usage, detection and IP configuration function. Acts identical to GetLastClientError

Memory Allocation

• `typedef void (*`**`PTR_Allocate`**`)(char** pbuffer, unsigned long* psize)`
Pointer to a memory allocation function. The function has to be supplied by the application developer using AV SDK.
- Pbuffer – pointer to allocated memory, returned value
- Psize – input/output value, poinster to memory size requested by the DLL. After the memory allocation the application programmer should specify the actual value of memory that was allocated

If the allocated memory size is zero or less than requested the DLL will return an error with code 2, see AVErrorCodes.h.

For example,
```
        // disallow memory allocation of sizes larger than 640K
        Allocate(char** pbuf, unsigned long* psize)
        {
            if(*psize > 640000){
                *pbuf = 0;
                return;
            }
        }
```
The function pointed to by PTR_Allocate is called in the following two cases:

- If the buffer size supplied by the user is insufficient for the image currently being acquired from the camera;
- When the user accesses camera registers;

• `typedef void (*`**`PTR_Deallocate`**`)(char* buffer)`

Pointer to a memory de-allocation function. The application programmer must supply this function, to be invoked by the DLL.

The function pointed to by PTR_Deallocate is called in the following cases:

- Immediately after the user passes to the DLL the pointer to new image buffer (either using function pointed to by PTR_Allocate or by using SetClientBuffer). In this case the function pointed to by PTR_Deallocate is passed the pointer to the buffer that was supplied to the DLL **prior** to the buffer currently in use by the DLL;
- When DLL is unloaded from memory;

Note that calling SetClientBuffer does not lead to the invocation of the function pointed to by PTR_Deallocate.

• `void` **`SetAllocateFunction`**`(PTR_Allocate)`

Specifies the function that the client will use to allocate memory. See usage notes and example below.

NOTE: C# users are recommended to use default memory management to avoid pointer operations and unsafe compiling mode. Default can be activated by calling `SetAllocateFunction`(null).

• `void` **`SetDeallocateFunction`**`(PTR_Deallocate)`

Specifies the function that the client will use to deallocate the memory.

1. Functions SetAllocateFunction and SetDeallocateFunction have to be called prior to using the client.
2. If allocated image buffer is too small to contain the arriving image, client calls the function PTR_Allocate, using parameter size to indicate the requested buffer size. PTR_Allocate should set the buffer pointer and allocate image buffer of equal or larger size, than requested, returning the actual allocated buffer size in the parameter size.

Example:

```
void OnAllocate(unsigned char** buffer, unsigned long* size)
{
        *size *= 2;
        *buffer = new unsigned char[size];
}
SetAllocateFunction(OnAllocate);
```
NOTE: C# users are recommended to use default memory management to avoid pointer operations and unsafe compiling mode. Default is activated by calling `SetDeallocateFunction`(null).

Camera Detection and IP Configuration Functions

- `int` **`FindCameras`**`(unsigned* number, unsigned timeout)`

Refresh list of cameras on the network. Number is both input and output parameter. As an input number points to number of re-tries. A re-try is attempted when the camera does not answer for some reason or the reply does not reach the source. As an output parameter (once FindCameras returns control) number is the pointer to the count of AV cameras found on the network. Timeout is the period of time between retries in milli-seconds. The function will wait for camera to reply during this time.

- `int` **`GetCameras`**`(AV2000Addr* paddr, unsigned* psize)`

Obtain a list of detected cameras. FindCameras function must be called prior to using this function. Paddr points to an array of AV2000Addr structures. Pointer `psize` points to the size of array paddr. Its value is the number of structures, not size in bytes. Normally `&number` is assigned to `psize` where "`number`" is the same variable used by FindCameras function as output.

- `int` **`SetCameraIp`**`(AV2000Addr*)`

Set camera IP address. To use the function fill out mac and ip arguments of AV2000Addr.

- `extern "C" export_dll_function int` **`GetMac`**`(AV2000Addr*)`

Returns MAC address of the camera if IP address is known.

Example:

```
AV2000Addr addr;
// known IP address of the camera 169.254.1.10
addr.ip[0] = 169;
addr.ip[1]  = 254;
addr.ip[2] = 1;
addr.ip[3] = 10;
GetMac(&addr); //addr.mac – contains MAC address of the camera;
```

- `int` **`CheckCamera`**`(const char* ip)`

Checks availability of a camera given its IP. "0" indicates that camera is not accessible. A positive value indicates success. SetAllocateFunction() and SetDeallocateFunction() must be invoked before using this function.

- `int` **`UpdateVersion`**`(int number)`

Forces the client to read from the camera its firmware version and adapt its operation, so that backwards compatibility with older firmware versions is assured.

- `unsigned long` **`Model`**`(int number)`

Returns camera model:

  - 1300 for AV1300, AV1310
  - 2000 for AV2000
  - 2100 for AV2100, AV2110
  - 3100 for AV3100, AV3110
  - 3130 for AV3130
  - 5100 for AV5100, AV5110
  - 8360 for AV8360
  - 8180 for AV8180
  - 1305 for AV1305, AV1355
  - 2105 for AV2105, AV2155
  - 3105 for AV3105, AV3155
  - 5105 for AV5105, AV5155
  - 8365 for AV8365
  - 8185 for AV8185
  - 3135 for AV3135

- o 10005 for AV10005

- `unsigned long` **`Version`**`(int number)`

Reads firmware version of the camera.

- `unsigned long` **`Revision`**`(int number)`

Reads PCB revision of the camera.

- `unsigned long` **`Dome`**`(int number)`

Return 1 if camera model is
- o AV1355
- o AV2155
- o AV3155
- o AV5155

Otherwise return 0

- `unsigned long` **`Compact`**`(int number)`

Return 1 if camera model is
- o AV1310
- o AV2110
- o AV3110
- o AV5110

Otherwise return 0

Auto-Iris functions

- `int` **`IrisPresent`**`(int number)`

Returns 1 if auto-iris presents, 0 otherwise.

DayNight functions

- `unsinged long` **`DayNight`**`(int number)`

Returns 1 if mechanical IR switcher presents, 0 otherwise.

Camera parameters cpDAY_NIGHT_MODE, cpDAY_NIGHT_TRIGGER_NIGHT, and cpDAY_NIGHT_TRIGGER_DAY have been redefined to support single sensor cameras equipped with a day/night switch. These parameters can be set/read through **SetAV2000Parameter()** and **GetAV2000Parameter()** functions. Please read p.8 for details.

CopyRight Strings functions

- `int` **`InitializeCopyRightStrings`**`(int number, EXIF_STRING_ID id, unsigned char* str);`

Set camera make/software/copyright strings which will appear in JPEG header in EXIF format.

Example:
```
dll.pInitializeCopyRightStrings(1, CAMERA_MAKE_STRING, "Arecont Vision");
dll.pInitializeCopyRightStrings(1, CAMERA_MODEL_STRING, "AV2100");
dll.pInitializeCopyRightStrings(1, SOFTWARE_STRING, "AV SDK v2.6");
dll.pInitializeCopyRightStrings(1, COPYRIGHT_STRING, "Arecont Vision");
```

CAMERA_MAKE_STRING, SOFTWARE_STRING, and COPYRIGHT_STRING accept up to 64 bytes, CAMERA_MODEL_STRING accepts up to 32 bytes.

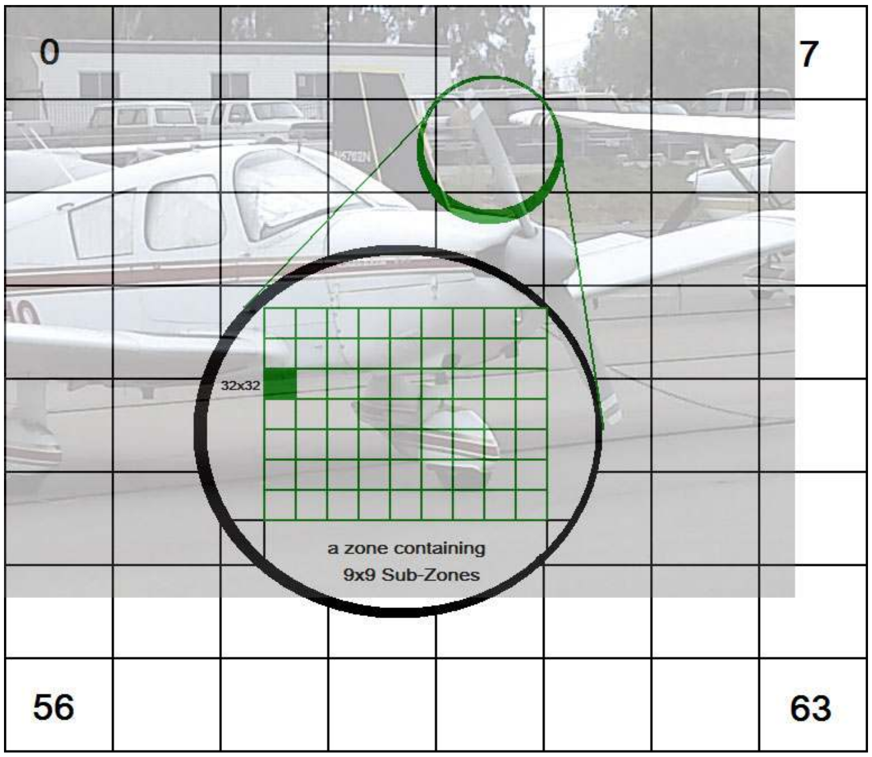
SDK will automatically generate a datetime_picture_taken field for each image in form of YYYY:MM:DD HH:MM:SS. The datetime information can be viewed in Windows.

SDK will calculate a checksum for each image and embed the checksum in the JPEG header. This field is normally invisible. Arecont Vision will provide a standalone utility to verify if the checksum matches current image content.

Motion Detection

On-camera motion detection is now supported by all AV camera models. The unique design of AV cameras allows supporting highly accurate 64-zone motion detection. To provide accurate motion detection in low contrast and low light environments, EACH pixel of EACH frame is analyzed. The user can set size of motion detection zones (via cpMD_ZONE_SIZE), select the sensitivity to motion (via cpMD_LEVEL_THRESH), select the zones where the motion detection has to be suspended (via cpMD_MATRIX) and decide what size of the moving objects is of interest (via cpMD_DETAIL). In addition to retrieving motion detection information, the camera can also be configured to output images only if motion is detected (via cpMD_MODE).

On-camera motion detection algorithm uses 8x8=64 square zones to cover the camera's sensor rectangle*. The zones are numbered from 0 to 63 as shown in the figure below. Each zone comprises a number of fixed-sized sub-zones with each sub-zone containing 32x32 pixels. Parameter cpMD_ZONE_SIZE specifies how many sub-zones (in one direction) a zone comprises. Developers should calcaute this parameter carefully so that the entire sensor rectangle can be covered by the zones.



*Camera's sensor rectangle is set using SDK functions SetAV2000Parameter() with parameter set {cpSENSOR_LEFT, cpSENSOR_TOP, cpSENSOR_WIDTH, cpSENSOR_HEIGHT} which are relative to the physical sensor dimension. HOWEVER, the actaul image size results from SDK functions such as GetWindowImage() DOES NOT necessarily equal the sensor retangle if cropping is explicitly applied.

For example, AV2100 camera's default sensor dimemsion is 1600x1200. The best-fit zone size will be cpMD_ZONE_SIZE = 7 so that 7x32x8 =1792 ≥ 1600. AV5100 camera's default sensor dimension is 2560x1600. The best-fit zone size will be cpMD_ZONE_SIZE = 11 so that 11x32x8=2816 ≥ 2560.

Important: users should keep it in mind that the total number of zones is always 64 (8 vertically and 8 horizontally). If the total zone size is larger than the acutal image size, some zones may not correspond to the active pixel array. In that case some motion detection values are not meaningful and should be ignored.

For example, for AV1300 camera: if the image size is 1280x1024, then for cpMD_ZONE_SIZE = 8 there are 5x4=20 active zones (3 zones after every 5 zones must be ignored during the GetMotionAray readout, as well as all zones after no. 31).

Parameter cpMD_DETAIL specifies at least how many sub-zones should contain motion for the zone in question to be considered as having motion. This parameter make it possible to filter out motions caused by objects smaller than those to be detected.

To get information on the motion within each zone the user should use the function `GetMotionArray`:

- `const unsigned char*` **`GetMotionArray`**`(int number)` - returns an array of motion detection data; number -- is the client number, the value returned is the pointer to a 64-byte array where the 0th byte corresponds to the upper left corner of the image, 63rd byte corresponds to the bottom right corner. The byte value indicates the number of sub-zones (blocks of 32x32 pixels) within each zone where motion was detected. The sub-zone size is fixed and cannot be changed.

Setting cpMD_MODE to '1' will cause the camera to output image only at the presence of motion. To exclude some zones from motion detection the user can use the parameter cpMD_MATRIX. To specify the active zones the user should call SetAV2000Parameter, passing the pointer to an 8-byte array, where each byte corresponds to one horizontal row of zones, with MSB corresponding to the left margin of the image and the LSB to the right. Any bit set to 0 disables motion detection in the corresponding zone.

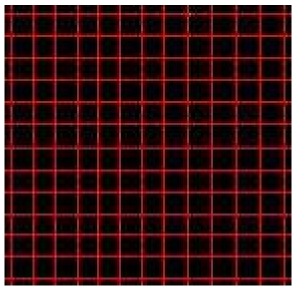
For example:

```
unsigned char arr[8];
      arr[0] = 0x7F;
for(int i = 1; i < 8; i++)
      arr[i] = 0xFF;
SetAV2000Parameter(1, cpMD_MATRIX, reinterpret_cast<long*>(arr));
```

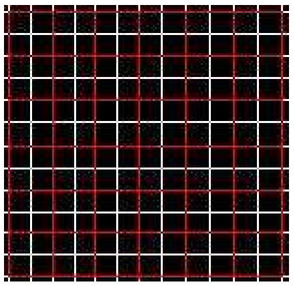
The above example sets all zones as active, with the exception of the upper leftmost zone. Note the use of the cast to (long *).

Binning Mode

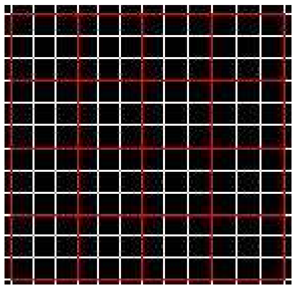
"Binning mode" is a feature supported by AV10005 cameras. When enabled, the image sensor is sampled in such a way that adjacent 2x2 pixels are combined into one effective superpixel. In binning mode the "superpixel" has larger area for light integration, thus the low light sensitivity is enhanced, ideally by a factor of 4. The figures below illustrate the general concept of binning mode. On AV1005 cameras only Binning 2x2 is supported.



Binning 1x1 (none)
pixels: 144



Binning 2x2
pixels: 36



Binning 3x3
pixels: 16

In SDK two parameters `cpBIN_DAY` and `cpBIN_NIGHT` are introduced to support binning mode. Parameter `cpBIN_DAY` controls binning when camera is operating in day mode, and `cpBIN_NIGHT` controls binning when camera is operating in night mode. Since it is normally during night time that low light sensitivity becomes a concern, the factory default values are chosen to be `cpBIN_DAY=0` (disabled) and `cpBIN_NIGIHT=1` (enabled) .

Note that binning mode results in lower image resolution. If the image request is `imFULL` resolution and requested size is 3648x2752, the actual size of returned image will be 1824x1376 if binning is enabled. Similarly, if the image request is `imHALF` resolution and requested size is 3648x2752, the actual size of returned image will be 912x688 if binning is enabled.

1080p Mode

"1080p mode" is a feature supported by AV10005 cameras, controlled by parameter `cpBIN_1080P` with default value 0 (disabled). When enabled, the 10 megapixel image sensor on AV10005 operates at 30fps in 1920x1080 resolution. When switching from 10meg mode to 1080p mode, applications should handle carefully the change of sensor size: the requested image size should be properly trimmed and centered to fit within 1920x1080 resolution, do NOT use the same request sizes for 10meg resolution images.

When motion detection is enabled, applications should also change the cpMD_ZONE_SIZE parameter to proper values to reflect the change of sensor size. The default zone size for 10meg resolution is 15, while the suitable zone size for 1080p mode is 8.

Single Capture Mode

"Single Capture mode" support is temporarily removed from latest camera firmware releases. Users who want to continue using this feature may need to downgrade firmware to older versions.

When single capture is activated, the camera stops sending live video and enters an endless loop to check for external trigger events. The software that communicates with the camera must send image requests constantly and check whether the response from the camera contains image. When there is no event, the camera responds to image requests with an empty data packet. When an event occurs, the camera responds to the most recent image request with the captured image.

Single capture function is primarily designed for low-light applications where an external flash is needed. When there is enough light, the camera automatically switches to a regular High Speed mode with short exposure time, about 1 ~ 2 ms, without triggering the flash.

To use the single capture function properly, the camera must be physically connected to an external trigger source (as input) and a flash (as output). Then the camera should go through a calibration process to setup a working environment. SDK users will use **SetCalibrateFlash**() function to initiate the calibration. The calibration process takes about 13 seconds, during which period the flash will be triggered 13 times. Once the calibration is done, users should check the result of calibration by calling **GetCalibrationNumber**() function. This function returns an integer resulting from the calibration. If the number is within **768 ~ 6144**, the calibration is successful, otherwise the calibration fails. For better image quality, one would expect the number to fall within **2000~3000** range. This number is under influence of the status of the iris. Closing the iris will cause the number to increase, and vice versa. Users may go through the calibration->adjusting iris->recalibration cycle for several times until the calibration result is satisfactory.

Once the calibration is done, the camera is ready to handle trigger events. During normal operation, there will be no image returned by functins like **GetImage2**(), **GetWindowImage**(), **GetDefaultImage**(), etc. In case of trigger, those functions will return a valid image. Users can judge the presence of image by the value of "size" paratmeter returned by those functions. If "size" is less than 1024, there is no image, otherwise the returned pointer points to a valid image.

For example:

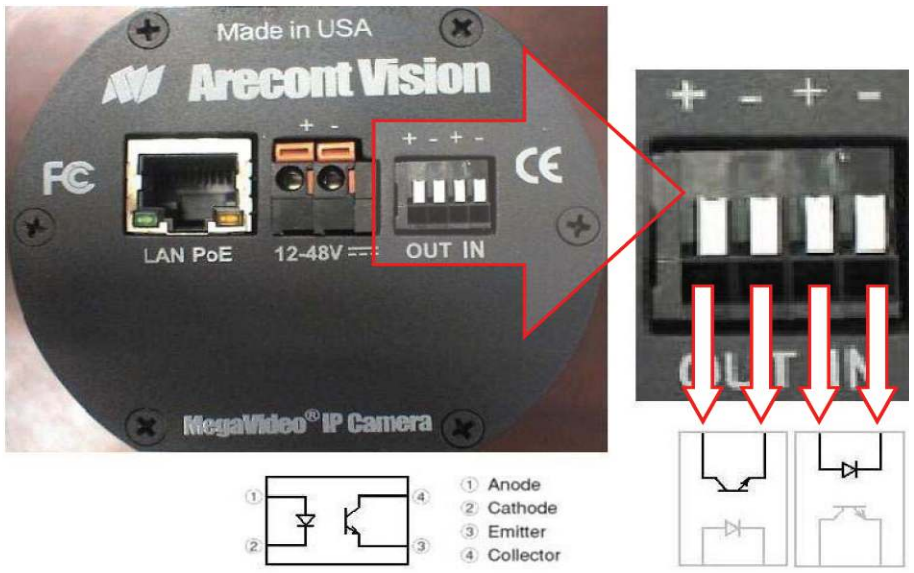
```
int client = 1, enable  = 1, ret = 0, ok=0, exit=0;
dll.pSetSingleCapture(client, enable);
dll.pSetCalibrateFlash(client);
Sleep(15000);
ret = dll.pGetCalibrationNumber(client);
ok = (ret > 768 && ret < 6144);

while(ok && !exit){
     GetWindowImage(client, buffer, size,…)
     if(size > 1024) // go on to process it
}
dll.pSetSingleCapture(client, enable=0);
```

Functon **TriggerSingleCapture**() is to initiate single capture from software. Users should make sure the camera firmware support this feature before trying to use this function.

Auxiliary IO Functions

The auxiliary input and output are accessible from the back panel, as shown below. Both the input and the output are electrically isolated from the camera's internal electrical circuitry by general-purpose photo couplers. The input is further protected with a serial 250 Ohm resistor and a debouncing circuit.



Electrical characteristics:

		Min	Max
Input voltage (V) (measured between + and – terminals)	ON	2.9	6.3
	OFF	0	1.3
Output current (mA) (measured between + and – terminals)	ON	-	50
	Collector-Emitter Voltage (V)	0	80
	OFF	-	0.1

The use of Auxilary IO functions through SDK involves three functions: **GetAuxIO**(), **SetAuxIO**() and **GetAuxOutStatus**().

GetAuxIO() function read status of IN port.
SetAuxIO() function set status of OUT port.
GetAuxOutStatus() function reads back current status of OUT port.

For example:

```
  //Get AuxIN status
  long client = 1, ret = 0;
  ret = dll.pGetAuxIO(client);  //ret: 1 if voltage present, 0 otherwise

  //Set AuxOUT status
  long curr_status = dll.pGetAuxOutStatus(client);
  long on_off = 1;   //on_off:  1 for short-circuit, 0 for open circuit
  dll.pSetAuxIO(client,on_off);
```

Caution: Starting from firmware version 64512 **GetAuxIO**() function reads AuxIN status from SDK local buffer instead of issuing new requests to camera. The purpose of this change is to improve efficiency. The SDK local buffer is updated when a new image arrives. For applications that do not request images until an event is detected on AuxIN port, it is now necessary to request a small resolution image before calling **GetAuxIO**() function.

Panoramic Camera Functions

The overall SDK usage for the panoramic cameras (AV8360, AV8180, AV8365, AV8185) is similar to other camera models SDK usage with some specific differences.

- The differences are related to the internal camera functionality. Each of the camera's four image sensors may be exposed to different lighting conditions, and, as a result, the sensors may run asynchronouosly to each other, their images arriving at different times, and/or at a different rate. The image arrival times and/or their relative rate may change at any time depending on the current lighting conditions. In order to avoid any idle time waiting for a particular requested image to arrive, the camera's hardware sends images on a "first in, first out" basis. For these reasons, the application cannot explicitly specify the channel number (sensor number) within each individual image request: it will receive the next available image from the camera. For the same reason, neither can the application specify any other image parameter (such as resolution, brightness, quality, etc.) within an individual image request. To get images from the panoramic camera the SDK supports only `GetDefaultImage,` while various image parameters are set separately in advance (see below).

- There are three kinds of images that can be requested from the panoramic camera: full resolution image, decimated image (1/4 size), and a zoom window image. After the application receives an image from the camera using **`GetDefaultImage`**(), it must determine which channel has been received, and whether the image is a zoom window.

    - To determine which channel the image was received from, use cpMS_NUMBER_OF_SENSOR.
    - To determine whether the recieved image was a zoom window use cpMS_IS_ZOOMED, similarly to the above (except for parameters).

- The application can set various parameters individually for each channel, as well as enable/disable full resolution and/or zoom wondows for each channel.

- When setting any parameter, specify which channel this setting applies to first, using cpMS_NUMBER_OF_SENSOR. For example, when setting brightness for channel 2, first set cpMS_NUMBER_OF_SENSOR = 2, then set brightness. The values for cpMS_NUMBER_OF_SENSOR range from 1 to 4.

- Image dimensions are set using cpSENSOR_LEFT, cpSENSOR_TOP, cpSENSOR_WIDTH, cpSENSOR_HEIGHT.

- A special camera mode is provided, in which each channel sends a full resolution image followed by four decimated images from all channels.. Typically, the decimated images are intended for displaying, while the full resolution images are intended for archival. It is enabled with cpMS_QUAD_MODE.

- Enabling/disabling camera channels is via cpMS_CHANNEL_ENABLE.

- Enabling/disabling full resolution channels is via cpMS_FULL_RES_ENABLE. The channels that are enabled via cpMS_CHANNEL_ENABLE, but not selected for full resolution via cpMS_FULL_RES_ENABLE send decimated images.

- Enabling/disabling zoom windows is via cpMS_ZOOM_WIN_ENABLE.

- In the panoramic camera motion detection is now supported. The usage of motion detection parameters (cpMD_ENABLED, cpMD_ZONE_SIZE, etc.) is similar to that of other models. The only tip is that cpMS_NUMBER_OF_SENSOR must be set properly before setting motion detection parameters.

For example:

```
// choose which sensor to set
long client = 1;
long num_of_sensor=1; //valid range 1~4
dll.pSetAV2000Parameter(client, cpMS_NUMBER_OF_SENSOR, &num_of_sensor);

//for the chosen sensor, enable/disable full resolution mode
long enable=1;
dll.pSetAV2000Parameter(client, cpMS_FULL_RES_ENABLE, &enable);

//for the chosen sensor, enable/disable that channel,
dll.pSetAV2000Parameter(client,cpMS_CHANNEL_ENABLE, &enable);

//request an image
dll.pGetDefaultImage(client, &Buffer, &Size, &Capacity);

//check the image comes from which sensor
if(Size > 1024)
dll.pGetAV2000Parameter(client, cpMS_NUMBER_OF_SENSOR, &num_of_sensor);

//check if it is zoom image, use only if zoom function is enabled
long is_zoom = 0;
dll.pGetAV2000Parameter(client, cpMS_IS_ZOOMED, &is_zoom);
```

To stream H.264 frames from AV8365/AV8185 cameras, the only option is to use GetDefaultImageEx() function defined as follows

```
•     extern "C"  int GetDefaultImageEx(int number, char** pdata,
unsigned long* size, unsigned long* capacity, int codec, int streamId,
int* Ifarme)
```

where;

- o pdata, size, capacity have the same meaning as in GetDefaultImage();
- o This function operates with cameras AV1300, AV2100 and AV3100, revision 5 and above, firmware version 52108 and above.

For example, the following code results in transmission of an H.264 frame:

```
    int num_of_sensor = 0;
    int is_zoom = 0;
    if (bRequestH264Image)
    {
        int Iframe = 0;
        int streamid = 1;
        ret = dll.pGetDefaultImageEx(1,
                                 &pcImageBuffer,&iImageBufferSize,
                                 &iImageBufferCapacity, H264_CODEC, streamid,
                                 &Iframe);
    }
    dll.pGetAV2000Parameter(1, cpMS_NUMBER_OF_SENSOR, &num_of_sensor);
    dll.pGetAV2000Parameter(1, cpMS_IS_ZOOMED, &is_zoom);
```

Note that the example code requests a P frame if possible. The actual frame type is determined by camera and returned in `Iframe`.

Examples

Arecont Vision provides a number of demo programs for different development tools and different model of cameras.

Development Tool	Package Name	Description
Borland C++ Builder 6.0	TestJpeg.rar	Support non-panoramic cameras, jpeg decompression, display and save to disk
	Test8360.rar	Support panoramic cameras, jpeg decompression, display and save to disk
Borland Delphi 7.0	DelphiRegualr	Support non-panoramic cameras, jpeg decompression, display and save to disk decompression uses Intel Jpeg Library (ijl15.dll)
Visual Studio 6.0 Console	Console.rar	Simple console application, find cameras and save image to disk
Visual Studio 6.0 MFC	MFC.rar	MFC example, jpeg decompression and display
Visual Studio 6.0 Win32	TestIJL.rar	Win32 application, display and save to disk, decompression uses Intel Jpeg Library (ijl11.dll)
Visual Studio 2005 C#	ApplicationRegular.rar	Support non-panoramic cameras, save to disk and display
	Application8360.rar	Support panoramic cameras, save to disk and display

DLL Usage Notes

Typical Usage

1. Create a client, CreateClient()
2. Set its IP, SetClientIP()
3. Receive camera version, UpdateVersion()
4. Main loop
   a. Set camera parameters
   b. Receive images, GetImage()
   c. Etc.
5. Release client, DestroyClient()

Multi-Threading Applications

1. Calling CreateClient() and DestroyClient() from different threads requires the use of critical sections or other synchronization mechnisms.

2. Synchronization is also required if the client with the same number (ID) is called from different threads. Invoking functions of different instances of the client from different threads does not require critical sections/ synchronization.

3. To obtain maximum possible frame rate client should be able to acknowledge arriving from the camera packets as soon as possible. Therefore, whenever possible, client should have a higher priority than other threads.

Miscellaneous

1. All structures used in the library are aligned at 4 byte boundary

2. Certain network equipment is not designed to sustain high speed communications and may create transmission problems or slow FPS. In particular, many 3COM routers are found to cause high packet loss thus are not in the list of recommendation.

Maximum Image Sizes Supported By AV Cameras

(in pixels)

Camera Model	Default Width	Default Height	Max Width	Max Height
AV1300/1310/1305/1355	1280	1024	1280	1024
AV2100/2110/2105/2155	1600	1200	1600	1200
AV3100/3110/3105/3155	1920	1200	2048	1536
AV3130/3135 – color	1920	1200	2048	1536
AV3130/3135 – mono	1280	1024	1280	1024
AV5100/5110/5105/5155	2560	1600	2592	1944
AV10005	3648	2752	3648	2752
AV8360/8365	1600 *	1200 *	1600 *	1200 *
AV8180/8185	1600 *	1200 *	1600 *	1200 *

* : per channel, each AV8360/8180 has four independent channels

Revision History

Version 2.8.2

- Added DayNight support for panoramic cameras AV836X, AV818X with a DayNight switcher.
- Added support and documentation for camera model AV10005.
- Added Dome() function to identify AV1355/2155/3155/5155 models.
- Added Compact() function to identify AV1310/2110/3110/5110 models.
- Added comments for GetAuxIO() function.

Version 2.7.6

- Added support for camera model AV3135.
- Added GetAuxOutStatus() function.

- Fixed issue with cpLIGHTING and cpSHORT_EXPOSURES parameters for panoramic models.

Version 2.7.4
- Fixed an error in this document about cpDAY_NIGHT_MODE parameter.
- Fixed issue with reading/setting DayNight parameters, affecting single sensor DN models.
- Fixed issue with reading cpGAMMA parameter.
- Fixed issue with reading cpLIGHTING and cpSHORT_EXPOSURES parameters.

Version 2.7.3
- Added GetDefaultImageEx() function to support H.264 panoramic cameras.
- Added motion detection support for panoramic cameras.
- Added cpGAMMA parameter supported by all models (may need firmware/hardware update)
- Extended range of cpSHORT_EXPOSURES parameter to [1..80] with default 5
- Fixed an issue that AV51xx models lose 50/60Hz settings during camera initialization.

Version 2.6.5
- Fixed an issue that GetMac() function does not support panoramic cameras.
- Fixed an issue that FindCameras() function does not reliably find cameras in busy network due to extended packet delay.
- Fixed an issue that EXIF header only supports one camera model string.

Version 2.6.4
- Fixed an issue with panoramic cameras that AV2000SDK.dll allows setting iris related parameters, thus overwriting color balance registers.

Version 2.6.3
- Redefined DayNight camera parameters to support cameras equipped with a day/night IR switch.
- Added InitializeCopyRightStrings() to support custom information.
- Added invisible image fingerprint in JPEG header for data integrity purposes.

Version 2.6.2
- Added GetWindowImageQEx() to support H.264 cameras
- Added demo for H.264 developers using MS C++ and C#

Version 2.5.40
- Added TriggerSingleCapture()
- Modified GetWindowImageQ() to support in-request doublescan

Version 2.5.39
- Added GetSingleCapture(),SetSingleCapture(),SetCalibrateFlash(),GetCalibrationNumber()
- Added SetAuxIO() and GetAuxIO()
- Added GetWindowImageQ()
- Added demo for Borland Delphi and MS C#
- Fixed a typo related to the description of cpSHORT_EXPOSURES
- Rewrote motion detection section

Version 2.5.26
- Added HIGH_SPEED and MOON_LIGHT™ modes for cpCAMERA_MODE
- Added cpSHORT_EXPOSURES for specifying fixed shutter widths

Version 2.5.24
- AddedAV8360 panoramic camera parameters and their description.
- Added parameters cpHEIGHT and cpWIDTH to allow obtaining larger than default images for AV3100, AV5100 and AV3130 when GetImage or GetImage2 is used.

Version 2.5.15
- Added motion detection parameter `cpMD_DETAIL.`

Version 2.5.14
- Fixed window size control for monochrome sensor of AV3130
- Added section on Motion Detection to the manual `cpMD_DETAIL`

Version 2.4.4
- Added support for the motion detection

Version 2.3.12
- Added checks for NULL pointers that might be erroneously passed by user to the SDK functions

- Commented out some motion detection related functions that are not yet used by the SDK but cause errors in some compilers

Version 2.3.11
- Modified default value of the parameter cpMD_LEVEL_THRESH (Rev5 cameras only)

Version 2.3.10
- Added error handling in the event of invocations of functions Model(int), Version(int), Revision(int) without prior call or failed call to UpdateVersion(int). Prior to this modification the resulting exceprion was not handled, now 0 is returned.
- Added some functions for support of on-camera motion detection. These functions are not yet complete and can only be used for testing.

Version 2.3.6
- Put a safeguard to prevent the overflow of sensor exposure register for AV2100 (exposure value can not exceed 14 bits)

Version 2.3.5
- Modified GetLastClientError function tocatch an error (return 0) in the event of a function call with non-existent client number.

Version 2.3.3
- Added parameters: cpREQUESTED_BLOCK_SIZE for camera with firmware versions of 52109 and above.

Version 2.3.1
- Added parameters: cpSENSOR_BLACK_WHITE_LEFT, cpSENSOR_BLACK_WHITE_TOP, cpSENSOR_BLACK_WHITE_WIDTH, cpSENSOR_BLACK_WHITE_HEIGHT, cpPER_CENT_IMAGE_RECTANGLE, cpREQUEST_LEFT, cpREQUEST_TOP, cpREQUEST_WIDTH, cpREQUEST_HEIGHT, cpQUALITY, cpRESOLUTION, cpSENSOR_LEFT, cpSENSOR_TOP, cpSENSOR_WIDTH, cpSENSOR_HEIGHT
- Added functions: GetWindowImage, GetMac and GetDefaultImage.
- Added table of maximum image sizes for AV cameras;
- Added support for 2x packet size – 2904 bytes for camera versions 52109 and above;

Version 2.2.19
- Added parameters: cpEXPOSURE_MODE, cpEXPOSURE_WINDOW_LEFT, cpEXPOSURE_WINDOW_TOP, cpEXPOSURE_WINDOW_WIDTH, cpEXPOSURE_WINDOW_HEIGHT

Version 2.2.12
- Added support for AV3130

Version 2.2.9
- Corrected functions SetCustomMode and GetCustomMode, added *pmax_digital_gain parameter and expanded the description

Version 2.2.6
- Added functions SetCustomMode and GetCustomMode
- Added value 3 for cpCAMERA_MODE

Version 2.2.4
- Added error code TFTP_MISSING_PACKETS to AVErrorCodes.h
- Expanded description of functions pointed to by PTR_ALLOCATE and PTR_Deallocate

Version 2.2.0
- Added function GetImage2

Version 2.0.14b
- cpBRIGHTNESS is -99..99

Version 2.0.14
- Corrected cpCAMERA_MODE values

Version 2.0.13
- SDK removed cpIRIS_REPOSITION_ENABLED (merged into cpIRIS_ENABLED)

Version 2.0.12
- SDK fixed cpIRIS_ENABLED setting
- SDK changed cpIRIS_REPOSITION_F_STOPS range to 1..15

- SDK changed cpIRIS_REPOSITION_F_STOPS_MIN range to 1..15
- SDK enlarged cpBRIGHTNESS range to -75..75
- SDK added cpAUTO_EXPOSITION, 0..1
- SDK changed some defaults

Version 2.0.11c
- Compression quality settings limited to 21

Version 2.0.11b
- Zoom description in GetImage()

Version 2.0.11
- Added int CheckCamera()
- Added cpDAY_NIGHT*

Version 2.0.10
- Added enum IRIS_STATUS
- Added GetIrisStatus()
- Updated cpROLL for camera versions before 51719
- Reduced camera communication access delay

Version 2.0.9g
- Added parameter support for camera version 50412

Version 2.0.9f
- cpIRIS_ENABLED, cpIRIS_REPOSITION_ENABLED values are 0 and non-zero

Version 2.0.9d
- IRIS_SPEED is 1..255, not 0..255

Version 2.0.9c
- Corrected formula and defaults for cpIRIS_REPOSITION_PERIOD, cpIRIS_REPOSITION_STABLE_PERIOD. cpDOUBLESCAN default is 0

Version 2.0.9b
- cpIRIS* attributes

Version 2.0.9
- Support storage of saturation, sharpness and rotation defaults on the camera. Firmware 50412 and later

Version 2.0.8
- Added FactoryDefault()
- Fixed GetAV2000Parameter for cpROLL, cpLIGHTING and cpCAMERA_MODE

Version 1.34
- SetAV2000Parameter() example in sample.cpp

Version 1.33
- Fixed description of ranges of brightness, blue/red adjustment, quality full, half, zoom
- CreateClient() changed, returns 0 when no error
- Added MaxClients()

Version 1.22
- Added AVErrorCodes.h
- Added SetClientTimeout()
- Added GetClientTimeout()
- Example to limit memory allocation size