ApBase API Development Guide

Documentation for the ApBase API

Introduction

The ApBase API provides an easy to use interface for controlling Aptina demo kit cameras.

ApBase performs these primary functions:

- Enumerate all of the demo kit cameras that are connected to the host PC.
- Run initialization scripts start the camera or change modes.
- Provide read/write access to sensor/SOC/ISP registers. Further, ApBase understands what register settings on the sensor do, keeps track of all relevant sensor modes, and automatically updates the demo kit hardware when the sensor mode changes. For example, if a register write enables MIPI output from the sensor, ApBase will automatically enable the MIPI receiver on the demo kit hardware.
- Convert the image received from the demo kit into RGB for saving to a file or displaying on the screen.
- Display images from the demo kit in a window.

Table of Contents

Quick Start

To use ApBase in C or C++, include the apbase.h header file, and link to apbase.lib.

Here is a minimal procedure to open a demo kit camera, initialize the Aptina device, grab an image and convert it to RGB.

```
#include “apbase.h”

AP_HANDLE hApBase = ap_Create(0);
if (hApBase == NULL)
   return;

ap_LoadIniPreset(hApBase, NULL, NULL);

size_t bufSize = ap_GrabFrame(hApBase, NULL, 0);
unsigned char *pBuffer = new unsigned char [bufSize];
ap_GrabFrame(hApBase, pBuffer, bufSize);

unsigned char *pRGB = NULL;
ap_u32 rgbWidth = 0, rgbHeight = 0, rgbBitDepth = 0;
if (ap_GetLastError() == MI_CAMERA_SUCCESS)
   pRGB = ap_ColorPipe(hApBase, pBuffer, bufSize, &rgbWidth, &rgbHeight,
                  &rgbBitDepth);

delete [] pBuffer;
ap_Destroy(hApBase);
ap_Finalize();
```

Note: The call to ap_LoadIniPreset() with NULL parameters is only applicable in situations where the default initialization is sufficient. For demo kits with multiple possible configurations, such as ISP + sensor combinations or different connection options, you will need to specify the particular initialization preset for the configuration.

Functions for Enumerating and Opening Devices

To access the camera, first create a device handle using a Create call, or a Probe call followed by a Create call.

ap_s32 ap_DeviceProbe(const char *szSensorDataDirOrFile)

Parameters

A pointer to a file name or directory name. Can be NULL.

Return Value

Error code.

Remarks

Search the system for Aptina demo kit devices or other compatible camera devices, identify the type of sensor or ISP, and build a set of internal data structures for each device. Use ap_Create() to create the handle used by the other API functions to access the device.

If the parameter is NULL, the default installation directory is searched for a sensor data file matching each device. Otherwise the parameter can be a specific sensor data file (.xsdat file) to be used with the device, or a directory to be searched for compatible sensor data files.

ApBase will keep track of the modes and state of all of the devices. When the application is done using the ApBase API, call ap_Finalize() to close all drivers and free all memory associated with the devices.

void ap_CancelProbe()

Cancel a device probe in progress in another thread.

int ap_NumCameras()

Return Value

Number of cameras.

Remarks

Return the number of cameras discovered by the device probe function.

void ap_Finalize()

Close all drivers and free all memory associated with the devices. Any existing AP_HANDLEs become invalid, so destroy all handles first.

ap_s32 ap_GetLastError()

Return Value

Error code.

Remarks

Return the error code from the last function call made in the current thread.

AP_HANDLE ap_Create(int nDeviceIndex)

Parameters

A device index, 0…ap_NumCameras() – 1.

Return Value

A device handle to be used with other ApBase functions.

Remarks

Create a handle to one of the devices discovered by the device probe. If the device probe has not been done yet, this function will call ap_DeviceProbeA() with the default parameters.

This handle is used by the rest of the API calls to specify which device that function should apply to. Creating and destroying these handles does not affect the device or the device state information kept within ApBase.

void ap_Destroy(AP_HANDLE apbase)

Parameters

Device handle.

Remarks

Delete a handle created by ap_Create(). All handles should be destroyed before calling ap_Finalize().

AP_HANDLE ap_CreateFromImageFile(const char *szFilename)

Parameters

A pointer to an image file name. Cannot be NULL.

Return Value

Device handle.

Remarks

Create a device handle from an image file. This is a way to make use of the image file reading feature of ApBase. Calling ap_GrabFrame() on this handle will retrieve the image. The ap_ColorPipe() call can convert it to RGB if it is not RGB already, for example a Bayer image or YCbCr image.

Note: Although it's possible to pass an image file name to ap_DeviceProbe() and call ap_Create(0) to get a handle, that method then precludes the possibility of probing for physical camera devices. This function is the preferred way to make a device handle from an image file.

Destroying this handle frees all associated memory and data structures.

AP_HANDLE ap_CreateVirtual(const char *szFilename)

Parameters

A pointer to a sensor data (.xsdat) file name. Cannot be NULL.

Return Value

Device handle.

Remarks

Create a handle from a sensor data (.xsdat) file without connecting to a physical camera; just create the data structures in memory. This is a way to make use of the xsdat file parser in ApBase. Functions like mi_FindRegister() and mi_FindRegisterAddr() can be used to translate register symbolic names to addresses and vice-versa. I/O functions will do nothing.

Destroying this handle frees all associated memory and data structures.

AP_HANDLE ap_CreateFromMidlibCamera(mi_camera_t *pCamera)

Parameters

A Midlib camera structure pointer, as returned from mi_OpenCameras(). Cannot be NULL.

Return Value

Device handle.

Remarks

Create an ApBase handle from a pre-existing Midlib camera structure pointer. This can be useful to make an ApBase call in an application that originally used only Midlib calls, and has already done a device probe with Midlib mi_OpenCameras().

mi_camera_t *ap_GetMidlibCamera(AP_HANDLE apbase)

Parameters

Device handle.

Return Value

Pointer to a Midlib camera structure.

Remarks

Retrieve a pointer to the Midlib camera structure for this device. This can be used to make Midlib calls to the device.

Note: Midlib can be used to write sensor registers or change the state of the camera device, but that can cause the state of the sensor to get out of sync with the state kept by ApBase. In that case use ap_CheckSensorState() to re-sync.

ap_u32 ap_GetMode(AP_HANDLE apbase, const char *szMode)

Parameters

apbase – Device handle.

szMode – Pointer to a mode name.

Return Value

The mode value.

Remarks

Get the current value of a demo kit device mode. The szMode strings are the same as the Midlib mi_modes symbols, without the MI_ prefix. For example “RX_TYPE”. These modes affect the demo kit boards, not the sensor. Use ap_GetLastError() to get the error code.

ap_s32 ap_SetMode(AP_HANDLE apbase, const char *szMode, ap_u32 nValue)

Parameters

apbase – Device handle.

szMode – Pointer to a mode name.

nValue – New value for the mode.

Return Value

Error code.

Remarks

Set the current value of a demo kit device mode. The szMode strings are the same as the Midlib mi_modes symbols, without the MI_ prefix. For example “RX_TYPE”. These modes affect the demo kit boards, not the sensor. Use ap_GetLastError() to get the error code.

Functions for the Debug Log

ApBase can log I/O activity and other info to a text file, which can be useful for debugging, or understanding what ApBase is doing at a low level. Your application can also output messages to the log.

void ap_OpenIoLog(ap_u32 nLogFlags, const char *szFilename)

Parameters

nLogFlags – A combination of AP_LOG_x flags.

szFilename – Name of a file for the log data. A new file is created each time this function is called. Digits 0, 1, etc. will be appended to the file name.

Remarks

Starts a new log file. The file extension is usually .txt. Digits are appended to the filename so the previous log is not overwritten when the program is run again. Use ap_GetIoLogFilename() to get the actual log filename if needed.

void ap_CloseIoLog()

Remarks

Close the current I/O log file.

const char * ap_GetIoLogFilename()

Return Value

The current I/O log file name.

Remarks

This function can be called after ap_CloseIoLog(), and will return the most recent log filename.

void ap_IoLogMsg(const char *szText)

Parameters

szText – A text string to add to the log.

Remarks

The text will be added to the log if there is a log file currently open and AP_LOG was one of the flags passed to OpenIoLog(). A newline at the end of the string is not needed. The text will be preceded by a timestamp in the log file.

void ap_IoLogDebugMsg(const char *szText, const char *szSource, const char *szFunc, ap_u32 nLine)

Parameters

szText – The message for the log.

szSource – Name of the current source file. Can be NULL.

szFunc – Name of the calling function.

nLine – Line number of the source file.

Remarks

The text will be added to the log if there is a log file currently open and AP_LOG_DEBUG was one of the flags passed to OpenIoLog(). A newline at the end of the string is not needed. The text will be preceded by a timestamp in the log file.

The szSource, szFunc, and nLine parameters are optional; if included they will be appended to the message. They are included to make it easy to find the location in your source code where a debug message originated from. Many compilers have macros like __FILE__, __FUNCTION__, __LINE__ that can be used here.

Functions for Executing Scripts

Scripts for initializing cameras and setting camera modes are contained in INI files.

ap_s32 ap_LoadIniPreset(AP_HANDLE apbase, const char *szIniFile, const char *szPreset)

Parameters

apbase – Device handle.

szIniFile – Pointer to an ini file name. Can be NULL.

szPreset – Name of the preset to run. Can be NULL.

Return Value

Error code.

Remarks

Execute the ini file preset specified. If the szIniFile parameter is NULL, the default ini file is used. If the szPreset parameter is NULL, the default initialization procedure is executed, usually "Python:", if it exists, followed by "Demo Initialization".

Note: Ini files with Python scripts may expect the preset named "Python:" to be executed once prior to any other preset in the file. ApBase will execute it if szPreset is NULL, but otherwise does not enforce this. The application should ensure that the "Python:" preset gets executed.

ap_s32 ap_CheckSensorState(AP_HANDLE apbase , unsigned int nCheckFlags)

Parameters

apbase – Device handle.

nCheckFlags – Not used, pass zero.

Return Value

Error code.

Remarks

Re-read all registers on the device to determine the current modes, and update the internal data structures and the demo kit hardware.

Although ApBase examines all register writes as they happen to track the current modes of the camera, there are cases where the internal state of ApBase can get out of sync with the device. For example, downloading a firmware patch to an SOC or ISP can have results that can't be predicted by the software. This function can be used to re-sync ApBase to the device state.

Functions for Reading Sensor or ISP Registers

.

ap_s32 ap_GetSensorRegister(AP_HANDLE apbase, const char *szRegisterName, const char *szBitfieldName, ap_u32 *pnValue, ap_u32 bCached)

Parameters

apbase – Device handle.

szRegisterName – Register name, from xsdat file.

szBitfieldName – Bitfield name, from xsdat file. Can be NULL to read the whole register.

pnValue – Pointer to memory to receive the value from the register.

bCached – If non-zero, use a cached register value if available.

Return Value

Error code.

Remarks

Read the contents of a sensor or ISP register or bitfield, referenced by symbolic name.

ap_s32 ap_GetSensorRegisterAddr(AP_HANDLE apbase, mi_addr_type nAddrType, ap_u32 nAddrSpace, ap_u32 nAddr, ap_u32 nDataBits, ap_u32 *pnValue, ap_u32 bCached)

Parameters

apbase – Device handle.

nAddrType – Address type symbol (MI_REG_ADDR, etc.)

nAddrSpace – Register page or addressable region.

nAddr – Register address.

nDataBits – Width of the register read operation.

pnValue – Pointer to memory to receive the value from the register.

bCached – If non-zero, use a cached value if available.

Return Value

Error code.

Remarks

Read the contents of a sensor or ISP register, referenced by register type and address.

Functions for Writing Sensor or ISP Registers

Register write flags

Some register write functions can return status flags.

AP_FLAG_OK – No side-effect.

AP_FLAG_REALLOC – Image format (width, height) or buffer size may change.

AP_FLAG_PAUSE – Sensor may stop streaming.

AP_FLAG_RESUME – Sensor may resume streaming.

AP_FLAG_NOT_SUPPORTED – Selecting a mode that is not supported by the demo system.

AP_FLAG_ILLEGAL_REG_COMBO – The new value creates an invalid combination with some other register(s).

AP_FLAG_ILLEGAL_REG_VALUE – The new value is not supported by the device.

AP_FLAG_REGISTER_RESET – Many other register values will change (reset or state change).

AP_FLAG_CLOCK_FREQUENCY – The clock frequency will change (this is a PLL register or clock divider).

AP_FLAG_REG_LIST_CHANGED – The set of camera registers will change.

AP_FLAG_NOT_ACCESSIBLE – Register is not accessible (standby).

AP_FLAG_READONLY – Writing a read-only register.

AP_FLAG_WRITEONLY – Reading a write-only register.

ap_s32 ap_SetSensorRegister(AP_HANDLE apbase, const char *szRegisterName, const char *szBitfieldName, ap_u32 nValue, ap_s32 *pnSideEffects)

Parameters

apbase – Device handle.

szRegisterName – Register name, from xsdat file.

szBitfieldName – Bitfield name, from xsdat file. Can be NULL to write the whole register.

nValue – New register value.

pnSideEffects – Pointer to a memory location to receive the side effects flags. Can be NULL.

Return Value

Error code.

Remarks

Write the contents of a sensor or ISP register or bitfield, referenced by symbolic name.

The pnSideEffects parameter will be set with flags indicating side-effects of writing the register. This parameter can be NULL.

ap_s32 ap_SetSensorRegisterAddr(AP_HANDLE apbase, mi_addr_type nAddrType, ap_u32 nAddrSpace, ap_u32 nAddr, ap_u32 nDataBits, ap_u32 nValue, ap_s32 *pnSideEffects)

Parameters

apbase – Device handle.

nAddrType – Address type symbol (MI_REG_ADDR, etc.)

nAddrSpace – Register page or addressable region.

nAddr – Register address.

nDataBits – Width of the register read operation.

nValue – New value for the register.

pnSideEffects – Pointer to a memory location to receive the side effects flags. Can be NULL.

Return Value

Error code.

Remarks

Same as ap_SetSensorRegisterA(), but the register is referenced by register type and address, and always writes the whole register.

ap_s32 ap_TestSetSensorRegister(AP_HANDLE apbase, const char *szRegisterName, ap_u32 nNewValue, ap_u32 nOldValue, ap_u32 *pnOldValue)

Parameters

apbase – Device handle.

szRegisterName – Register name, from xsdat file.

nNewValue – New register value.

nOldValue – Current register value.

pnOldValue – Pointer to a memory location to receive the current register value. May be NULL.

Return Value

A combination of side-effects flags.

Remarks

Get the side-effects of a register write, without writing the register; the register being referenced by symbolic name. This lets the application know if a register write will have side-effects or an error before writing the register. For example, if there is an error flag, the application could skip the register write, or prompt the user.

If pnOldValue is non-NULL, ap_TestSetSensorRegister() will read the current value of the register from the device. If the current value of the register is already known to the caller, pass it in nOldValue and pass NULL for pnOldValue. Otherwise pass a pointer to a variable as pnOldValue.

If the side-effects flags include AP_FLAG_REALLOC then the image format would change, and background threads streaming images should be stopped before the register is written.

ap_s32 ap_ TestSetSensorRegisterAddr(AP_HANDLE apbase, mi_addr_type nAddrType, ap_u32 nAddrSpace, ap_u32 nAddr, ap_u32 nDataBits, ap_u32 nNewValue, ap_u32 nOldValue, ap_u32 *pnOldValue)

Parameters

apbase – Device handle.

nAddrType – Address type symbol (MI_REG_ADDR, etc.)

nAddrSpace – Register page or addressable region.

nAddr – Register address.

nDataBits – Width of the register read operation.

nNewValue – New value for the register.

nOldValue – Current register value.

pnOldValue – Pointer to a memory location to receive the current register value. May be NULL.

Return Value

A combination of side-effects flags.

Remarks

Same as ap_TestSetSensorRegisterA(), but the register referenced by address type and address.

ap_s32 ap_GetLastSideEffects()

Return Value

Combination of side-effects flags.

Remarks

Returns the side-effects flags from the last sensor register-set function called in the current thread.

Functions for Enumerating Sensor, ISP or Demo Kit Registers

These functions enable enumerating all of the named registers in a .xsdat file or .cdat file, for example to create a register dump or populate a GUI.

ap_s32 ap_NumRegisters(AP_HANDLE apbase, const char *szDevice)

Parameters

apbase – Device handle.

szDevice – Part number of sensor or ISP, or chip name of Demo kit FPGA. Can be NULL or empty, which will refer to the sensor or ISP.

Return Value

Number of registers on the device, or 0 on error.

Remarks

This function can also be used to test for the existence of specific demo kit hardware.

AP_HANDLE ap_Register(AP_HANDLE apbase, const char *szDevice, int nIndex)

Parameters

apbase – Device handle.

szDevice – Part number of sensor or ISP, or chip name of Demo kit FPGA. Can be NULL or empty, which will refer to the sensor or ISP.

nIndex – Ordinal of a register, 0 to ap_NumRegisters() – 1.

Return Value

A register object handle for the register property functions.

Remarks

Use ap_DestroyRegister() to free the handle.

AP_HANDLE ap_FindRegister(AP_HANDLE apbase, const char *szDevice, const char *szRegister)

Parameters

apbase – Device handle.

szDevice – Part number of sensor or ISP, or chip name of Demo kit FPGA. Can be NULL or empty, which will refer to the sensor or ISP.

szRegister – Symbolic name of a register.

Return Value

A register object handle for the register property functions.

Remarks

Use ap_DestroyRegister() to free the handle.

void ap_DestroyRegister(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Remarks

Free the handle.

const char * ap_RegisterSymbol(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

Symbolic name of the register.

Remarks

This name is used by other API calls, in INI file scripts, etc.

const char * ap_RegisterDisplayName(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

Display name of the register.

Remarks

Very often this is the same as the symbolic name, but lower case.

const char * ap_RegisterDetail(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

Short description of the register.

Remarks

A one-line description of the register.

const char * ap_RegisterLongDesc(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

Long description of the register.

Remarks

This is the full data sheet documentation of the register.

ap_u32 ap_RegisterBitwidth(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

The width of the register in bits.

Remarks

Can be 8, 16 or 32. In many cases, only a subset of the bits are meaningful.

ap_u32 ap_RegisterMask(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

Which bits of the register are meaningful.

Remarks

.

ap_u32 ap_RegisterDefault(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

The default value of the register.

Remarks

.

const char * ap_RegisterDatatype(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

Data type of the register.

Remarks

Example values are “unsigned”, “signed”, “fixed8”, etc.

double ap_RegisterMinimum(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

The minimum recommended or useful value of the register.

Remarks

The return value is converted to floating point according to the data type of the register.

double ap_RegisterMaximum(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

The maximum recommended or useful value of the register.

Remarks

The return value is converted to floating point according to the data type of the register.

ap_u32 ap_RegisterAddr(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

The register address.

Remarks

.

ap_u32 ap_RegisterAddrSpace(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

The address space value of the register.

Remarks

.

const char * ap_RegisterAddrSpaceId(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

Symbolic name of the register's address space.

Remarks

.

const char * ap_RegisterAddrSpaceName(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

Display name of the register's address space.

Remarks

.

ap_s32 ap_RegisterAddrType(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

The address type of the register.

Remarks

Possible values are 0 for simple register, 1 for firmware variable, 2 for advanced register other RAM address, 3 for other indirectly addressed register, or 4 for attached sensor register (as on an ISP with an attached sensor).

ap_u32 ap_RegisterRw(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

The allowed access to the register.

Remarks

Possible values are 0 for read/write, 1 for read-only or 2 for write-only.

Functions for Enumerating Sensor, ISP or Demo Kit Registers

These functions enable enumerating all of the named bitfields for a register a defined in the .xsdat file or .cdat file, for example to create a register dump or populate a GUI.

ap_s32 ap_NumBitfields(AP_HANDLE apreg)

Parameters

apreg – Register handle.

Return Value

Number of bitfields defined for the given register. Returns 0 on error.

Remarks

.

AP_HANDLE ap_Bitfield(AP_HANDLE apreg, int nIndex)

Parameters

apreg – Register handle.

nIndex – Ordinal of a bitfield, 0 to ap_NumBitfields() – 1.

Return Value

A bitfield object handle for the bitfield property functions.

Remarks

Use ap_DestroyBitfield() to free the handle.

AP_HANDLE ap_FindBitfield(AP_HANDLE apbase, const char *szDevice, const char *szRegister, const char szBitfield)

Parameters

apbase – Device handle.

szDevice – Part number of sensor or ISP, or chip name of Demo kit FPGA. Can be NULL or empty, which will refer to the sensor or ISP.

szRegister – Symbolic name of a register.

szBitfield – Symbolic name of a bitfield.

Return Value

A bitfield object handle for the bitfield property functions.

Remarks

Use ap_DestroyBitfield() to free the handle.

AP_HANDLE ap_BitfieldByName(AP_HANDLE apreg, const char szBitfield)

Parameters

apreg – Register object handle.

szBitfield – Symbolic name of a bitfield.

Return Value

A bitfield object handle for the bitfield property functions.

Remarks

Use ap_DestroyBitfield() to free the handle.

void ap_DestroyBitfield(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Remarks

Free the handle.

const char * ap_BitfieldSymbol(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Return Value

Symbolic name of the bitfield.

Remarks

This name is used by other API calls, in INI file scripts, etc.

const char * ap_BitfieldDisplayName(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Return Value

Display name of the bitfield.

Remarks

Very often this is the same as the symbolic name, but lower case.

const char * ap_BitfieldDetail(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Return Value

Short description of the bitfield.

Remarks

A one-line description of the bitfield.

const char * ap_BitfieldLongDesc(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Return Value

Long description of the bitfield.

Remarks

This is the full data sheet documentation of the bitfield.

ap_u32 ap_BitfieldBitwidth(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Return Value

The width of the bitfield in bits.

Remarks

Can be any value from 1 up to the bitwidth of the parent register.

ap_u32 ap_BitfieldMask(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Return Value

Which bits of the parent register belong to this bitfield.

Remarks

.

ap_u32 ap_BitfieldAdjustedMask(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Return Value

The bitfield bitmask shifted right to the LSB.

Remarks

This mask is suitable for masking off a value to be written to the bitfield.

ap_u32 ap_BitfieldDefault(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Return Value

The default value of the bitfield.

Remarks

.

const char * ap_BitfieldDatatype(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Return Value

Data type of the bitfield.

Remarks

Example values are “unsigned”, “signed”, “fixed8”, etc.

double ap_BitfieldMinimum(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Return Value

The minimum recommended or useful value of the bitfield.

Remarks

The return value is converted to floating point according to the data type of the bitfield.

double ap_BitfieldMaximum(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Return Value

The maximum recommended or useful value of the bitfield.

Remarks

The return value is converted to floating point according to the data type of the bitfield.

ap_u32 ap_BitfieldRw(AP_HANDLE apbitf)

Parameters

apbitf – Bitfield handle.

Return Value

The allowed access to the bitfield.

Remarks

Possible values are 0 for read/write, 1 for read-only or 2 for write-only.

Functions for Getting and Processing Images

.

ap_s32 ap_GetImageFormat(AP_HANDLE apbase, ap_u32 *pWidth, ap_u32 *pHeight, char *szImageType, ap_u32 nBufferSize)

Parameters

apbase – Device handle.

pWidth – Pointer to a variable to receive the current width. Can be NULL.

nHeight – Pointer to a variable to receive the current height. Can be NULL.

szImageType – Buffer to receive the name of the current image type. Can be NULL.

nBufferSize – Size of the szImageType buffer.

Return Value

1 if the current device settings match a standard image type, 0 otherwise.

Remarks

Retrieves the current image format of the device. The possible image types are "BAYER-6", "BAYER-8", "BAYER-10", "BAYER-8+2", "BAYER-10IHDR", "BAYER-12", "BAYER-12HDR", "BAYER-8+4", "BAYER-14", "BAYER-14HDR", "BAYER-16", "BAYER-20", "BAYER-16+4", "BAYER-STEREO", "YCBCR", "YCBCR-16", "YCBCR-10", "YCBCR-20", "YUV-420", "M420", "RGB-565", "RGB-555", "RGB-444X", "RGB-X444", "RGB-332", "RGB-24", "RGB-32", "RGB-48", "BGRG", "JPEG", "JPEG-SPEEDTAGS", and "JPEG-ROT".

ap_s32 ap_SetImageFormat(AP_HANDLE apbase, ap_u32 nWidth, ap_u32 nHeight, const char *szImageType)

Parameters

apbase – Device handle.

nWidth – New image width, or 0 to keep the current width.

nHeight – New image height, or 0 to keep the current height.

szImageType – Name of new image type, or NULL to keep the current image type.

Return Value

Error code.

Remarks

In order for ap_GrabFrame() and ap_ColorPipe() to work correctly, ApBase must know the image format coming out of the camera. ApBase normally calculates the image format from the device register settings automatically. This function can be used as an override if necessary.

Note: This function does not change the camera mode or settings. To change the camera mode you need to write registers or run ini scripts (which write registers).

ap_u32 ap_GrabFrame(AP_HANDLE apbase, unsigned char *pBuffer, ap_u32 nBufferSize)

Parameters

apbase – Device handle.

pBuffer – Pointer to a memory buffer large enough for the device driver to receive an entire image.

nBufferSize – Size of allocated memory in bytes.

Return Value

Number of bytes of image data, or buffer size if pBuffer is NULL.

Remarks

Get the next image from the camera into the specified buffer. The caller must allocate the buffer. Calling this function with NULL pBuffer returns the minimum required buffer size.

Returns 0 on error. Call ap_GetLastError() for the error code.

unsigned char *ap_ColorPipe(AP_HANDLE apbase, unsigned char *pInBuffer, ap_u32 nInBufferSize, ap_u32 *rgbWidth, ap_u32 *rgbHeight, ap_u32 *rgbBitDepth)

Parameters

apbase – Device handle.

pInBuffer – Pointer to a memory buffer holding an image received from the device, or a similarly formatted image.

nInBufferSize – Size of pInBuffer in bytes.

rgbWidth – Pointer to a memory location to receive the width of the output image. May be NULL.

rgbHeight – Pointer to a memory location to receive the height of the output image. May be NULL.

rgbBitDepth – Pointer to a memory location to receive the pixel size of the output image. May be NULL.

Return Value

A pointer to a memory buffer containing the output image. NULL if there was an error.

Remarks

Convert image data returned from ap_GrabFrame(), or similarly formatted image, to RGB. The output is normally 32 bits per pixel with B – G – R – X byte order. The output image could also be either 24 or 16 bits per pixel if an output window has been set, and the display is 24 or 16 bits per pixel. The output will match the display format. See ap_SetWindow().

The converted image could conceivably have different dimensions than the input image. The rgbWidth, rgbHeight and rgbBitDepth parameters will be filled in with the converted image size. These parameters may be NULL.

The caller should not free the returned buffer. The memory may be re-used on subsequent calls to ap_ColorPipe(), so the data must be copied to another location to be kept.

int ap_GetState(AP_HANDLE apbase, const char *szState)

Parameters

apbase – Device handle.

szState – Name of an integer-valued color pipe state variable.

Return Value

Value of the variable.

Remarks

Get the value of an integer-valued color pipe state variable. See the INI File User's Guide for a list of color pipe variables.

void ap_SetState(AP_HANDLE apbase, const char *szState, int nValue)

Parameters

apbase – Device handle.

szState – Name of an integer-valued color pipe state variable.

nValue – New value for the variable.

Remarks

Set the value of an integer-valued color pipe state variable.

const char *ap_GetStateStr(AP_HANDLE apbase, const char *szState)

Parameters

apbase – Device handle.

szState – Name of a string-valued color pipe state variable.

Return Value

Value of the variable.

Remarks

Get the value of a string-valued color pipe state variable. The memory pointed to by the returned pointer may be reused on subsequent calls to ap_GetStateStrA(), so the application must copy the data to keep it.

void ap_SetStateStr(AP_HANDLE apbase, const char *szState, const char *szValue)

Parameters

apbase – Device handle.

szState – Name of an string-valued color pipe state variable.

szValue – New value for the variable.

Remarks

Set the value of a string-valued color pipe state variable.

ap_s32 ap_SetWindow(AP_HANDLE apbase, void *hWnd, int nViewWidth, int nViewHeight, int bFullScreen)

Parameters

apbase – Device handle.

hWnd – Native window handle, or NULL.

nViewWidth – Width to display the image.

nViewHeight – Height to display the image.

bFullScreen – If non-zero use full screen mode.

Return Value

Error code.

Remarks

Set a window handle of a window where ap_ColorPipe() should draw the converted RGB image. To un-set the window call this function with hWnd = NULL.

void ap_RecordStart(AP_HANDLE apbase)

Parameters

apbase – Device handle.

Remarks

Henceforth, ap_ColorPipe() will record each image processed into a video file.

Note: At this time, this recording RAW video files is not recommended because the library does not yet create the _info.txt file.

void ap_RecordStop(AP_HANDLE apbase)

Parameters

apbase – Device handle.

Remarks

Henceforth, ap_ColorPipe() will no longer record images.

Setting Callback Functions

These functions set callbacks so the application can handle operations that may need user interaction or are otherwise appropriate to be handled at the application level.

The pContext parameter should point to an application-owned object containing the data that will be needed by the callback function. If pContext is NULL the apbase handle will be passed to the application.

void ap_SetCallback_BeginAccessRegs(AP_HANDLE apbase, AP_BeginAccessRegs_PROC pProc, void *pContext)

Parameters

apbase – Device handle.

pProc – Pointer to a callback function with the following parameters and return value:

```
void APBASE_DECL MyBeginAccessRegs(void *pContext) { … }
```

pContext – Pointer to an application-defined data structure that will be passed unchanged to the callback. May be NULL, in which case apbase will be passed to the callback.

Remarks

ApBase is about to read or write registers.

void ap_SetCallback_EndAccessRegs(AP_HANDLE apbase, AP_EndAccessRegs_PROC pProc, void *pContext)

Parameters

apbase – Device handle.

pProc – Pointer to a callback function with the following parameters and return value:

```
void APBASE_DECL MyEndAccessRegs(void *pContext) { … }
```

pContext – Pointer to an application-defined data structure that will be passed unchanged to the callback. May be NULL, in which case apbase will be passed to the callback.

Remarks

ApBase has finished accessing registers.

void ap_SetCallback_GetSaveDir(AP_HANDLE apbase, AP_GetSaveDir_PROC pProc, void *pContext)

Parameters

apbase – Device handle.

pProc – Pointer to a callback function with the following parameters and return value:

```
void APBASE_DECL MyGetSaveDir(void *pContext, char *szBuffer, int
                    nBufferLen) { … }
```

szBuffer – Pointer to a buffer to hold the directory name.

nBufferLen – Length of the buffer in bytes.

pContext – Pointer to an application-defined data structure that will be passed unchanged to the callback. May be NULL, in which case apbase will be passed to the callback.

Remarks

The SAVE_IMAGE= or SAVE_REGS= ini command is querying which directory to save the files into. Fill in the buffer with the directory path.

void ap_SetCallback_SetOption(AP_HANDLE apbase, AP_SetOption_PROC pProc, void *pContext)

Parameters

apbase – Device handle.

pProc – Pointer to a callback function with the following parameters and return value:

```
void APBASE_DECL MySetOption(void *pContext, const char *szOption, const
                    char *szValue, int nValue) { … }
```

szOption – Name of the option.

szValue – If non-NULL, value of the option as a string.

nValue – If szValue is NULL, value of the option as an integer.

pContext – Pointer to an application-defined data structure that will be passed unchanged to the callback. May be NULL, in which case apbase will be passed to the callback.

Remarks

Handler for the OPTION= ini command. If szValue is NULL this is an integer-valued option, otherwise it is a string-valued option. This provides a mechanism for scripts to set application variables or invoke application functions.

void ap_SetCallback_MultipleChoice(AP_HANDLE apbase, AP_MultipleChoice_PROC pProc, void *pContext)

Parameters

apbase – Device handle.

pProc – Pointer to a callback function with the following parameters and return value:

```
int APBASE_DECL MyMultipleChoice(void *pContext, const char *szMessage,
                    const char *szChoices) { … }
```

szMessage – Prompt message.

szChoices – Sequence of choice strings.

pContext – Pointer to an application-defined data structure that will be passed unchanged to the callback. May be NULL, in which case apbase will be passed to the callback.

Remarks

Handler for the PROMPT= ini command. The handler should present the choices to the user and return which option was selected. The szMessage parameter is a prompt string. szChoices is a sequence of NUL-terminated strings representing the choices. The end of the sequence is denoted by a zero-length string (two NULs).

The callback function should return 1 for the first choice, 2 for the second choice, etc.; or 0 for no choice or -1 for Cancel.

If no MultipleChoice callback has been provided, the PROMPT ini command will default to the first choice.

void ap_SetCallback_LogComment(AP_HANDLE apbase, AP_LogComment_PROC pProc, void *pContext)

Parameters

apbase – Device handle.

pProc – Pointer to a callback function with the following parameters and return value:

```
void APBASE_DECL MyLogComment(void *pContext, const char *szComment) { … }
```

szComment – Log string.

pContext – Pointer to an application-defined data structure that will be passed unchanged to the callback. May be NULL, in which case apbase will be passed to the callback.

Remarks

Handler for the LOG= ini command. Show the string to the user, for example in a Log window or on the console.

void ap_SetCallback_LogDebugTrace(AP_HANDLE apbase, AP_LogDebugTrace_PROC pProc, void *pContext)

Parameters

apbase – Device handle.

pProc – Pointer to a callback function with the following parameters and return value:

```
int APBASE_DECL MyLogDebugTrace(void *pContext, const char *szTrace) { … }
```

szTrace – Trace string.

pContext – Pointer to an application-defined data structure that will be passed unchanged to the callback. May be NULL, in which case apbase will be passed to the callback.

Remarks

Handler for ini debug trace messages. This shows each ini file command as it is executed, for debugging ini files. Similar handling as for LogComment(). Do nothing if szTrace is NULL.

The handler should return 0 if debug messages are not desired, or 1 if they are desired.

void ap_SetCallback_ErrorMessage(AP_HANDLE apbase, AP_ErrorMessage_PROC pProc, void *pContext)

Parameters

apbase – Device handle.

pProc – Pointer to a callback function with the following parameters and return value:

```
int APBASE_DECL MyErrorMessage(void *pContext, const char *szMessage,
                    unsigned int mbType) { … }
```

szMessage – Message string.

mbType – Message type. See Windows MessageBox().

pContext – Pointer to an application-defined data structure that will be passed unchanged to the callback. May be NULL, in which case apbase will be passed to the callback.

Remarks

Handler for error messages from ApBase, usually ini syntax errors or runtime errors such as attempting to write a read-only register. The handler could be implemented with a MessageBox, or output in some kind of error log. The mbType parameter is suitable for use with the Windows MessageBox() call.

The handler should return 1 for OK, or 2 for Cancel.

void ap_SetCallback_ScriptOutput(AP_HANDLE apbase, AP_ScriptOutput_PROC pProc, void *pContext)

Parameters

apbase – Device handle.

pProc – Pointer to a callback function with the following parameters and return value:

```
void APBASE_DECL MyScriptOutput(void *pContext, int nUnit, const char
                    *szString) { … }
```

nUnit – Output destination. 0 for stdout; 1 for stderr.

szString – Output text.

pContext – Pointer to an application-defined data structure that will be passed unchanged to the callback. May be NULL, in which case apbase will be passed to the callback.

Remarks

Handler for Python print() calls. The nUnit parameter is 0 for stdout, or 1 for stderr. Show the string to the user, such as in a window, or output to the application's stdout or stderr.

void ap_SetCallback_ScriptInput(AP_HANDLE apbase, AP_ScriptInput_PROC pProc, void *pContext)

Parameters

apbase – Device handle.

pProc – Pointer to a callback function with the following parameters and return value:

```
const char * APBASE_DECL MyScriptInput(void *pContext, int nUnit) { … }
```

nUnit – Always 0.

pContext – Pointer to an application-defined data structure that will be passed unchanged to the callback. May be NULL, in which case apbase will be passed to the callback.

Remarks

Handler for Python input() calls. The handler should return a string representing the user's input. This function may need to allocate memory for the string. If so, also implement ScriptInputFree().

void ap_SetCallback_ScriptInputFree(AP_HANDLE apbase, AP_ScriptInputFree_PROC pProc, void *pContext)

Parameters

apbase – Device handle.

pProc – Pointer to a callback function with the following parameters and return value:

```
void APBASE_DECL MyScriptInputFree(void *pContext, const char *szString) { …
                    }
```

szString – Pointer returned from MyScriptInput().

pContext – Pointer to an application-defined data structure that will be passed unchanged to the callback. May be NULL, in which case apbase will be passed to the callback.

Remarks

Hook to free any memory that may have been allocated by the ScriptInput() handler. The szString parameter is the same pointer that was returned from the ScriptInput() handler.