(C) 2018 WINDGO Inc.
FILE: Î¼JPEG - C Header
NAME: ujpeg.h
DATE: 2018/01/12
TIME: 11:00:01
 *
// uJPEG (MicroJPEG) -- KeyJ's Small Baseline JPEG Decoder
// based on NanoJPEG -- KeyJ's Tiny Baseline JPEG Decoder
// version 1.2 (2012-02-19)
// by Martin J. Fiedler <martin.fiedler@gmx.net>
//
// This software is published under the terms of KeyJ's Research License,
// version 0.2. Usage of this software is subject to the following conditions:
// 0. There's no warranty whatsoever. The author(s) of this software can not
//   be held liable for any damages that occur when using this software.
// 1. This software may be used freely for both non-commercial and commercial
//   purposes.
// 2. This software may be redistributed freely as long as no fees are charged
//   for the distribution and this license information is included.
// 3. This software may be modified freely except for this license information,
//   which must not be changed in any way.
// 4. If anything other than configuration, indentation or comments have been
//   altered in the code, the original author(s) must receive a copy of the
//   modified code.
//
//
//
// This is a minimal decoder for baseline JPEG images. It accepts JPEG files as
// input and generates either 8-bit grayscale or packed 24-bit RGB images as
// output. It does not fully parse JFIF or Exif headers; all JPEG files
// are assumed to be either grayscale or YCbCr. CMYK or other color spaces are
// not supported. All YCbCr subsampling schemes with power-of-two ratios are
// supported, as are restart intervals. Progressive or lossless JPEG is not
// supported.
// Summed up, uJPEG should be able to decode all images from digital cameras
// and most common forms of other non-progressive JPEG images.
//
// The decoder is not optimized for speed, it's optimized for simplicity and
// small code. Image quality should be at a reasonable level. A bicubic chroma
// upsampling filter ensures that subsampled YCbCr images are rendered in
// decent quality. If Exif information is present, uJPEG automatically detects
// whether centered or co-sited chroma sample positioning is used.
//
// The decoder offers only a minimal amount of error resilience. If anything is
// wrong with the image's headers, it will reject the complete image. If
// there's an error within the coefficient bitstream, the decoder will stop
// decoding at the error position and leave the remainder of the image gray.
//
// The code should work with every modern C compiler without problems and
// should not emit any warnings. It uses only (at least) 32-bit integer
// arithmetic and is supposed to be endianness independent, 64-bit clean and
// thread-safe. It uses no external libraries except the C runtime (libc).

```
#ifndef _UJPEG_H_
#define _UJPEG_H_

// result codes for ujDecode()
typedef enum _uj_result {
    UJ_OK           = 0,  // no error, decoding successful
    UJ_NO_CONTEXT   = 1,  // called uj* function without image handle
    UJ_NOT_DECODED  = 2,  // image has not yet been decoded
    UJ_INVALID_ARG  = 3,  // invalid argument
```

```
    UJ_IO_ERROR     = 4,  // file I/O error
    UJ_OUT_OF_MEM   = 5,  // out of memory
    UJ_NO_JPEG      = 6,  // not a JPEG file
    UJ_UNSUPPORTED  = 7,  // unsupported format
    UJ_SYNTAX_ERROR = 8,  // syntax error
    UJ_INTERNAL_ERR = 9,  // internal error
    __UJ_FINISHED         // used internally, will never be reported
} ujResult;

// plane (color component) structure
typedef struct _uj_plane {
    int width;              // visible width
    int height;             // visible height
    int stride;             // line size in bytes
    unsigned char *pixels;  // pixel data
} ujPlane;


//////////////////////////////////////////////////////////////////////////////
// C INTERFACE                                                              //
//////////////////////////////////////////////////////////////////////////////

#ifdef __cplusplus
extern "C" {
#endif

// data type for uJPEG image handles
typedef void* ujImage;

// return the error code of the last uJPEG operation
extern ujResult ujGetError(void);

// create a uJPEG image context
extern ujImage ujCreate(void);

// tell the context not to decode image data (only parse headers)
extern void ujDisableDecoding(ujImage img);

// tell the context whether which chroma upsampling mode to use
#define UJ_CHROMA_MODE_FAST      1  // low-quality pixel repetition (fast)
#define UJ_CHROMA_MODE_ACCURATE  0  // accurate bicubic upsampling (slower)
#define UJ_CHROMA_MODE_DEFAULT   0  // default mode: accurate
extern void ujSetChromaMode(ujImage img, int mode);

// decode a JPEG image from memory
// img:  the handle to the uJPEG image to decode to;
//       if it is NULL, a new instance will be created
// jpeg: a pointer to the JPEG image file in memory
// size: the size of the JPEG image file in memory
// returns the image handle on success or NULL on failure; use ujGetError to
// get a more detailed error description
extern ujImage ujDecode(ujImage img, const void* jpeg, const int size);

// decode a JPEG image from a file
// img:  the handle to the uJPEG image to decode to;
//       if it is NULL, a new instance will be created
// filename: the name of the file to load
// returns the image handle on success or NULL on failure; use ujGetError to
// get a more detailed error description
extern ujImage ujDecodeFile(ujImage img, const char* filename);

// determine whether a picture has been successfully decoded
extern int ujIsValid(ujImage img);
```

```
// determine the dimensions of a decoded picture
extern int ujGetWidth(ujImage img);
extern int ujGetHeight(ujImage img);

// determine whether a decoded picture is grayscale (0) or color (1)
extern int ujIsColor(ujImage img);

// determine the amount of memory required to hold a decoded and converted
// picture
extern int ujGetImageSize(ujImage img);

// retrieve a pointer to the internal buffer of a decoded plane
// num is the plane number: 0 = Y (luminance), 1 = Cb, 2 = Cr.
// returns a pointer or NULL in case of failure
extern ujPlane* ujGetPlane(ujImage img, int num);

// retrieve decoded and converted picture
// If called with dest == NULL, uJPEG will create an internal buffer to hold
// the decoded and converted picture and returns the pointer to this buffer.
// If called with dest != NULL, uJPEG will convert the image into a user-
// supplied buffer and return the address of that buffer.
// This function can be called with dest=NULL multiple times without performance
// penalty.
// If conversion failed, this function returns NULL; use ujGetError to get a
// more detailed error description.
extern unsigned char* ujGetImage(ujImage img, unsigned char* dest);

// destroy a uJPEG image handle
extern void ujDestroy(ujImage img);

// destroy a uJPEG image handle and make the handle variable unusable
// (preferred to ujDestroy)
#define ujFree(img) do { ujDestroy(img); img = NULL; } while (0)

#ifdef __cplusplus
}
#endif


//////////////////////////////////////////////////////////////////////////////
// C++ INTERFACE                                                            //
//////////////////////////////////////////////////////////////////////////////

#ifdef __cplusplus

class uJPEG {
public:
    uJPEG()                                     { img = ujCreate(); }
    virtual ~uJPEG()                            { ujFree(img); }
    static ujResult getError()                  { return ujGetError(); }
    void disableDecoding()                      { ujDisableDecoding(img); }
    void setChromaMode(int mode)                { ujSetChromaMode(img, mode); }
    bool decode(const void* jpeg, const int size) { return ujDecode(img, jpeg, size) !=
 NULL; }
    bool decodeFile(const char* filename)       { return ujDecodeFile(img, filename)
!= NULL; }
    bool isValid()                              { return (ujIsValid(img) != 0); }
    bool good()                                 { return  isValid(); }
    bool bad()                                  { return !isValid(); }
    int getWidth()                              { return ujGetWidth(img); }
    int getHeight()                             { return ujGetHeight(img); }
    bool isColor()                              { return (ujIsColor(img) != 0); }
```

```
    int getImageSize()                          { return ujGetImageSize(img); }
    ujPlane* getPlane(int num)                  { return ujGetPlane(img, num); }
    const unsigned char* getImage()             { return ujGetImage(img, NULL); }
    bool getImage(unsigned char* dest)          { return ujGetImage(img, dest) != NUL
L; }
private:
    ujImage img;
};

#endif//__cplusplus

#endif//_UJPEG_H_

 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
END OF FILE Î¼JPEG - C Header
NAME: ujpeg.h
DATE: 2018/01/12
TIME: 11:00:01
(C) 2018 WINDGO Inc.
```