Tapwave® TwJpg Graphics API Reference

Version 1.1a

Tapwave TwJpg Graphics API Reference

Copyright

Background

To use the TwJpg library, your application must include `TwJpg.h`, which is automatically included by `Tapwave.h`.

This library provides facilities for decoding "jpeg" images. The image data is delivered to the library through a callback called a "TwJpgImageReader". Because a callback is used, the data can be located in a resource, in memory, or in a file on a card. The image decoding can produce a buffer of pixel data in memory or a TwGfx surface. During the decoding process the image can also be scaled; this scaling can optionally be anti-aliased for a higher quality result.

Most of these calls will allocate temporary storage using the dynamic heap. If these allocations fail then an error of sysErrNoFreeRAM will be returned.

Library Data Types

```
typedef struct TwJpgImageTag TwJpgImageType;

typedef struct TwJpgImageInfoType {
    Int32 size;   /* caller MUST set this to sizeof(TwJpgImageInfoType) */

    Int32 width, height; /* image dimensions */
    Int32 colorspace;    /* colorspace */
    Int32 components;    /* # of color components (RGB==3, for example) */

    // TODO: add "standard" markers
} TwJpgImageInfoType;

/*
 * Image reader callback. This is used by the jpg functions to read
 * data for a given image. The return value is a count of the number
 * of bytes read. Zero indicates an EOF, positive values indicate the
 * amount returned. Negative values indicates an error. aHandle is a
 * data value provided to the callback for the usage of the callback
 * implementation.
 */
typedef Int32 (*TwJpgImageReader)(void* aHandle,
                                  void* aBuffer,
                                  UInt32 aAmount);

/*
 * This predicate is invoked during the decode process to see if the
 * decoder should abort the current decode. The predicate should
 * return a non-zero value if the decode should be aborted, zero
 * otherwise.
 */
typedef Int32 (*TwJpgAbortCheck)(void* aHandle);
```

API

TwJpgOpenImage

Purpose	Open an image for eventual decoding.	
Prototype	`Err TwJpgOpenImage(TwJpgImageType** aResult, TwJpgImageReader aReader, void* aHandle)`	
Parameters	`[out] aResult`	Pointer to a handle to the image. If the request succeeds then `*aResult` is set to a handle to the image for use in subsequent calls.
	`[in] aReader`	Pointer to a TwJpgImageReader function that is used by the library to provide data during the decoding process.
	`[in] aHandle`	An opaque pointer provided by the caller which is passed through to the reader callback function.
Result	`errNone` – Succeeded `TwJpgErrorNullPointer` –either aResult or aReader is NULL	
Header	`TwJpg.h`	

TwJpgCloseImage

Purpose	Release the resources associated with the image.	
Prototype	`Err TwJpgCloseImage(TwJpgImageType* aImage)`	
Parameters	`[in] aImage`	A handle to the image.
Result	`errNone` – Succeeded `twJpgErrorInvalidHandle` – the handle to the image was invalid	
Side Effects	This function releases all resources allocated by the TwJpg library for the image.	
Header	`TwJpg.h`	

TwJpgSetAbortCheck

Purpose	Set the “abort check” callback function associated with the image.	
Prototype	`Err TwJpgSetAbortCheck(TwJpgImageType* aImage, TwJpgAbortCheck aChecker, void* aHandle)`	
Parameters	`[in] aImage`	A handle to the image.
	`[in] aChecker`	A callback function called during image decoding to check if the decoding process should be aborted.
	`[in] aHandle`	An opaque pointer provided by the caller which is passed through to the checker callback function.
Result	`errNone` – Succeeded `twJpgErrorInvalidHandle` – the handle to the image was invalid	
Side Effects	During the image decoding process, if an abort check callback is defined, the callback will be invoked periodically to see if the decoding process should be aborted or not.	
Header	`TwJpg.h`	

TwJpgGetAbortCheck

Purpose	Get the “abort check” callback function associated with the image.	
Prototype	`Err TwJpgGetAbortCheck(TwJpgImageType* aImage, TwJpgAbortCheck* aCheckerResult, void** aHandleResult)`	
Parameters	`[in] aImage`	A handle to the image.
	`[out] aCheckerResult`	Out parameter used to store the last abort check callback function set by TwJpgSetAbortCheck. NULL will be stored if no call to TwJpgSetAbortCheck has been made.
	`[out] aHandleResult`	A pointer to the opaque handle that will be set to the value provided by the last call to TwJpgSetAbortCheck.
Result	`errNone` – Succeeded `twJpgErrorInvalidHandle` – the handle to the image was invalid `twJpgErrorNullPointer` – the aCheckerResult or aHandleResult pointers are NULL	
Side Effects	None.	
Header	`TwJpg.h`	

TwJpgGetImageInfo

Purpose	Get image information.	
Prototype	`Err TwJpgGetImageInfo(TwJpgImageType* aImage, TwJpgImageInfoType* aInfoResult)`	
Parameters	`[in] aImage`	A handle to the image.
	`[inout] aInfoResult`	In/Out parameter used to store the information about the image. This pointer must not be NULL and the size field must be initialized to the size of the data structure.
Result	`errNone` – Succeeded `twJpgErrorInvalidHandle` – the handle to the image was invalid `twJpgErrorNullPointer` – the aInfoResult is a null pointer `twJpgErrorBadObjectVersion` – the `TwJpgImageInfoType` size field doesn't match a known version for the library	

Side Effects	This call returns the dimensions of the image, as well as its colorspace and number of color components (e.g. 1 for grayscale, 3 for RGB). The TwJpgImageInfoType has the following fields:
	
	`typedef struct TwJpgImageInfoType {`
	`  Int32 size;`
	`  Int32 width, height;  /* image dimensions */`
	`  Int32 colorspace;  /* colorspace */`
	`  Int32 components;  /* # of color components (RGB==3) */`
	`} TwJpgImageInfoType;`
	
	Note that the "size" field must be set to "sizeof(TwJpgImageInfoType)" before the call is made otherwise twJpgErrorBadObjectVersion may be returned.
	
	Also, this call will read the "image header" from the jpeg data which means that the reader callback function set by the TwJpgOpenImage call.	

Header	TwJpg.h

TwJpgDecodeImage

Purpose	Fully decode the image into a buffer allocated by this call.	

Prototype	`Err TwJpgDecodeImage(TwJpgImageType* aImage,`
	`Int32 aPixelFormat,`
	`void** aBufferResult)`	

Parameters	`[in] aImage`	A handle to the image.
	`[in] aPixelFormat`	This argument describes the desired format of the decoded pixel data.
	`[out] aBufferResult`	A pointer to the pixel buffer result. This pointer is set to the buffer allocated by the decoder.
Result	`errNone` – Succeeded `twJpgErrorInvalidHandle` – the handle to the image was invalid `twJpgErrorNullPointer` – aBufferResult is a null pointer `twJpgErrorInvalidPixelFormat` – aPixelFormat is invalid `twJpgErrorBadImage` – the image decoding failed	
Side Effects	Decode the entire image into *aBufferResult. TwJpgDecodeImage will allocate the memory and return it in *aBufferResult. The caller must free it when done using MemPtrFree. The format of the buffer data is determined by the aPixelFormat argument. Note that this is the only API that will provide RGB888 bits. The buffer data will contain the image width times the bytes per pixel (2 or 3) for each row of data. The first row of data corresponds to a “y” of zero. The second row of data (if there is one) corresponds to a “y” of one, and so on. This call will return twJpgErrorDecodeAborted if the abort checker aborts the decode. Note that it is allowed by this API to decode an image multiple times. It is the callers responsibility to rewind the reader state to allow for this to actually	

	work. If the caller does not rewind the reader state then most likely the subsequent call will yield a twJpgErrorBadImage error. If the image decode fails for any reason (premature “EOF” on the reader data, invalid data in the stream, etc) then the twJpgErrorBadImage will be returned.
Header	TwJpg.h

TwJpgDecodeAndScaleImage

Purpose	Fully decode and scale the image into a buffer allocated by this call.	
Prototype	`Err TwJpgDecodeAndScaleImage(TwJpgImageType* aImage, Int32 aScaledWidth, Int32 aScaledHeight, Boolean aAntiAlias, void** aBufferResult)`	
Parameters	`[in] aImage`	A handle to the image.
	`[in] aScaledWidth`	The desired width of the scaled result.
	`[in] aScaledHeight`	The desired height of the scaled result.
	`[in] aAntiAlias`	A flag indicating whether or not to perform an anti-aliased scale operation. See below for more information.
	`[out] aBufferResult`	A pointer to the pixel buffer result. This pointer is set to the buffer allocated by the decoder.
Result	`errNone` – Succeeded `twJpgErrorInvalidHandle` – the handle to the image was invalid `twJpgErrorNullPointer` – aBufferResult is a null pointer `twJpgErrorBadImage` – the image decoding failed `twJpgErrorInvalidSize` – the scaled width/height values are invalid	
Side Effects	The image is fully decoded with the same behavior as TwJpgDecodeImage with a pixel format of twJpgPixelFormatRGB565_LE. The scaling works as follows: preliminary scaling is done using the jpeg	

	decoding process. If possible, entire blocks of jpeg data will be skipped to produce a reduced size image (clearly this only applies only if the scaling is shrinking the image dimensions). This will speed up the decode time of large images as a side effect. If the decode level scaling doesn't produce the exact size image then a second level of scaling is done using the TwJpgScaleImageBuffer function. The aAntiAlias flag is passed into this routine to determine if point sampling or anti-aliasing is used for the scaling. The resulting buffer will have a bytes per row that is two times the scaled width and there will be aScaledHeight rows.
Header	`TwJpg.h`

TwJpgDecodeImageToSurface

Purpose	Decode the image into an existing TwGfx surface.	
Prototype	`Err TwJpgDecodeImageToSurface(TwJpgImageType* aImage, Boolean aAntiAlias, TwGfxSurfaceType* aDestSurface)`	
Parameters	`[in] aImage`	A handle to the image.
	`[in] aAntiAlias`	A flag indicating whether or not to perform an anti-aliased scale operation. See below for more information.
	`[in] aDestSurface`	The surface to decode the image into.
Result	`errNone` – Succeeded `twJpgErrorInvalidHandle` – the handle to the image was invalid `twJpgErrorBadImage` – the image decoding failed	
Side Effects	The image is fully decoded with the same behavior as TwJpgDecodeImage with a pixel format of twJpgPixelFormatRGB565_LE. The image will be scaled to fit the surface dimensions exactly as if by calling TwJpgScaleImageBuffer and passing in the aAntiAlias flag.	
Header	`TwJpg.h`	

TwJpgDecodeImageToNewSurface

Purpose	Decode the image into a new TwGfx surface.	
Prototype	`Err TwJpgDecodeImageToNewSurface(TwJpgImageType* aImage, Int32 aScaledWidth, Int32 aScaledHeight, Boolean aAntiAlias, TwGfxType* aGfxLib, TwGfxSurfaceType** aSurfaceResult)`	
Parameters	`[in] aImage`	A handle to the image.
	`[in] aScaledWidth`	The desired width of the scaled result.
	`[in] aScaledHeight`	The desired height of the scaled result.
	`[in] aAntiAlias`	A flag indicating whether or not to perform an anti-aliased scale operation. See below for more information.
	`[in] aGfxLib`	A handle to an open instance of the TwGfx library.
	`[out] aSurfaceResult`	A pointer to where the allocated surface handle will be stored.
Result	`errNone` – Succeeded `twJpgErrorInvalidHandle` – the handle to the image was invalid `twJpgErrorBadImage` – the image decoding failed `twJpgErrorInvalidSize` – the scaled width/height values are invalid	
Side	This call is simlar to TwJpgDecodeToSurface except that a new surface is created	

Effects	with the dimensions of aScaledWidth and aScaledHeight. The image will be scaled to fit the surface dimensions exactly as if by calling TwJpgScaleImageBuffer and passing in the aAntiAlias flag.
Header	TwJpg.h

TwJpgScaleImageBuffer

Purpose	Scale a block of image data that is in the pixel format of twJpgPixelFormatRGB565_LE.	
Prototype	`Err TwJpgScaleImageBuffer(void* aBufferIn, Int32 aInWidth, Int32 aInHeight, Int32 aScaledWidth, Int32 aScaledHeight, Boolean aAntiAlias, void** aBufferResult)`	
Parameters	`[in] aBufferIn`	A handle to the input pixel data.
	`[in] aInWidth`	The width, in pixels, of the input data. The bytes per row of the input data is two times the width in pixels.
	`[in] aInheight`	The height of the input image data.
	`[in] aScaledWidth`	The desired width of the scaled result.
	`[in] aScaledHeight`	The desired height of the scaled result.
	`[in] aAntiAlias`	A flag indicating whether or not to perform an anti-aliased scale operation.
	`[out] aBufferResult`	A pointer to the pixel buffer result. This pointer is set to the buffer allocated by the scaler.
Result	`errNone` – Succeeded `twJpgErrorInvalidHandle` – the handle to the image was invalid	

	`twJpgErrorBadImage` – the image decoding failed `twJpgErrorInvalidSize` – the scaled width/height values are invalid or the input width/height values are invalid.
Side Effects	This is a mostly general purpose scaling routine. If aAntiAlias is false then the image data is scaled using “point sampling”, otherwise a higher quality (but much slower) anti-aliased scaler is used. The input image data is restricted to be in the pixel format of twJpgPixelFormatRGB565_LE and must organized such that the bytes per row is the same as the input width times two.
Header	`TwJpg.h`

Examples

This first example uses VFS to read jpeg files from a card and decode them to a block of memory.

```
static Int32 vfsReader(void* aHandle, void* aBuffer, UInt32 aAmount) {
    FileRef ref = (FileRef) aHandle;
    UInt32 nbytes;
    Err err = VFSFileRead(ref, aAmount, aBuffer, &nbytes);
    return (err && err != vfsErrFileEOF) ? -err : nbytes;
}

{
    Err err;
    TwJpgImageType* img;
    void* pixels;

    /* not shown: open the file using VFSFileOpen */
    err = TwJpgOpenImage(&img, vfsReader, (void*)fileRef);
    err = TwJpgDecodeImage(img, twJpgPixelFormatRGB565_LE, &pixels);
}
```