Tapwave® TwGfx Graphics API Reference

Version 1.1a

Copyright

1.Background

To use the TwGfx library your application must include TwGfx.h, which is automatically included by Tapwave.h. The Tapwave ROM restricts access to the TwGfx API to only those applications that have been digitally signed. See the *Digital Rights Management* document for more information about signing your application.

You must open the library with TwGfxOpen before calling any other functions, and you must call TwGfxClose before your application exits. You can call TwGfxOpen as many times as desired, but you must pair each request with a matching TwGfxClose.

The library provides facilities for the allocation and manipulation of surfaces. Surfaces are rectangular regions of memory associated with the graphics accelerator. In addition to basic rendering operations (points, lines, rectangles) there are a large set of *bit blt* operations used to perform surface to surface copy operations. Note that bit blt operations that use the same surface for both the source and destination will yield undefined results in the overlapping area.

The library provides access to the Palm display surface (the surface used to refresh the TFT display) via the TwGfxGetPalmDisplaySurface API. This surface object remains consistent with the size, shape, location and orientation of the application drawing area, which excludes the Pen Input and Status areas. The Palm display surface size, shape, location and orientation are controlled by the PINSetInputAreaState, StatShow, StatHide and SysSetOrientation API's (see PenInputMgr.h for more information). In addition, calls to WinScreenLock are also tracked by the Palm display surface – rendering shifts to an offscreen surface during a lock operation, and is automatically blt'd to the onscreen surface when the lock ends.

Surface memory can be accessed directly by the CPU. However, this access is not as efficient as accessing CPU memory. To increase performance, Tapwave provides several *asynchronous* procedures for transferring data to/from a surface using background DMA. This allows the CPU to continue executing during a transfer operation. When such a transfer is started (see wGfxReadSurface

or wGfxWriteSurface

later in this document) the surface is considered *busy*. Any other attempt to use the surface will fail, returning a twGfxErrorOperationInProgress error. To determine when a surface is no longer busy use the TwGfxIsSurfaceReady procedure.

In addition to the asynchronous copy operation, there is also an *asynchronous bit blt* operation. This is used to avoid waiting for a vertical blank before issuing a bitblt, thus freeing up the CPU to perform other calculations. There can be at most one asynchronous bitblt going at any time, for all instances of the library.

2.Library data types

```
typedef struct TwGfxInfoType {
  /* requester MUST set this to sizeof (TwGfxInfoType) */
  Int32 size;

  Int32 displayWidth, displayHeight;    /* current dimensions of display */
  Int32 displayRowBytes;                /* byte width of entire row */
  Int32 displayPixelFormat;             /* format of display */

  Int32 freeAcceleratorMemory;          /* free accelerator memory */
  Int32 totalAcceleratorMemory;         /* total accelerator memory */
} TwGfxInfoType;

/*
 * Surface information structure
 */
typedef struct TwGfxSurfaceInfoType {
  /* requester MUST set this to sizeof(TwGfxSurfaceInfoType) */
  Int32 size;

  Int32 width, height;                /* dimensions */
  Int32 rowBytes;                     /* byte width of entire row */

  Int32 location;                     /* memory location of the surface */
  Int32 pixelFormat;                  /* format of the surface */
} TwGfxSurfaceInfoType;

/*
 * Bitmap structure
 */
typedef struct TwGfxBitmapType {
  /* requester MUST set this to sizeof(TwGfxBitmapType) */
  Int32 size;

  Int32 width, height;                 /* size of bitmap */
  Int32 rowBytes;                      /* bytes per row */
  Int32 pixelFormat;                   /* format of data */
  void* data;                          /* actual data */

  UInt16* displayPalette;              /* In native display format */
  TwGfxPackedRGBType transparentColor;
} TwGfxBitmapType;

typedef struct TwGfxPointType {
  Int32 x, y;
} TwGfxPointType;

typedef struct TwGfxRectType {
  Int32 x, y, w, h;
} TwGfxRectType;
```

```
typedef struct TwGfxSpanType {
  Int32 x, y, w;
} TwGfxSpanType;
```

3.Library macros

The first set of macros convert colors between the various color formats supported by the library. The format of all TwGfx surfaces is 16 bits per pixel (little endian) with an RGB value of 565. This means that there are 5 bits of red, 6 bits of green and 5 bits of blue. In general, the API functions do not define surface colors; the API is more generic (and upward compatible) because it accepts colors in "packed component RGB" format, which are simply RGB 888 (8 bits per color component).

Also note that the color conversions from RGB 888 to RGB 565 use simple mask and shift operations. No attempt is made to "round" or apply any other color policy to mitigate the difference in accuracy between the two color types. It is the responsibility of the application to fine-tune the conversion process.

```
/*
 * Macro to construct an rgb565 color from
 * three 8 bit components. This is a manually optimized version that
 * is consistent with the other color macros here.
 *
 * Note that the 68k version of this macro generates data that is
 * byte-swapped into little-endian format. This means that data passed
 * to TwGfxDrawBitmap and TwGfxWriteSurface will be properly arranged
 * if you use this macro.
 */
#define TwGfxMakeDisplayRGB_BigEndian(_r,_g,_b) \
  ( (((_g) & 0xFC) << 11) | (((_b) & 0xF8) << 5) | \
    ((_r) & 0xF8) | (((_g) & 0xFF) >> 5) )

#define TwGfxMakeDisplayRGB_LittleEndian(_r,_g,_b) \
  ( (((_r) & 0xF8) << 8) | (((_g) & 0xFC) << 3) | (((_b) & 0xF8) >> 3) )

#if CPU_TYPE == CPU_68K
#define TwGfxMakeDisplayRGB(_r,_g,_b) \
    TwGfxMakeDisplayRGB_BigEndian(_r,_g,_b)
#else
#define TwGfxMakeDisplayRGB(_r,_g,_b) \
    TwGfxMakeDisplayRGB_LittleEndian(_r,_g,_b)
#endif

/*
 * These macros take an 8-bit color component and adjust the
 * size to match the rgb565 display framebuffer
 */
#define TwGfxRComponentToDisplayComponent(_r) (((_r) & 0xF8) >> 3)
#define TwGfxGComponentToDisplayComponent(_g) (((_g) & 0xFC) >> 2)
#define TwGfxBComponentToDisplayComponent(_b) (((_b) & 0xF8) >> 3)

/*
 * This macro converts from packed component RGB to packed display RGB
 */
#define TwGfxPackedRGBToDisplayRGB(_rgb)                                \
  ( (TwGfxRComponentToDisplayComponent((_rgb) >> 16) << twGfxRShift) | \
```

```
    (TwGfxGComponentToDisplayComponent((_rgb) >> 8) << twGfxGShift) |  \
    (TwGfxBComponentToDisplayComponent((_rgb)) << twGfxBShift) )

/*
 * This macro converts a TwGfxRGBType structure to a
 * TwGfxPackedRGBType value.
 */
#define TwGfxRGBToPackedRGB(_rgb) \
  ((TwGfxPackedRGBType) ( ((_rgb).r << 16) | ((_rgb).g << 8) | (_rgb).b ) )

/*
 * This macro converts rgb components to a TwGfxPackedRGBType
 */
#define TwGfxComponentsToPackedRGB(_r,_g,_b)      \
  ((TwGfxPackedRGBType) ( (((_r) & 0xFF) << 16) | \
                          (((_g) & 0xFF) << 8) |  \
                          ((_b) & 0xFF) ) )
```

The following convenience macros can be used to simplify and clarify code.

```
/*
 * This macro helps fill in a TwGfxPointType
 */
#define TwGfxMakePoint(_point, _x, _y) \
  ((_point).x = (_x), (_point).y = (_y))

/*
 * This macro helps fill in a TwGfxRectType
 */
#define TwGfxMakeRect(_rect, _x, _y, _w, _h) \
  ((_rect).x = (_x), (_rect).y = (_y), (_rect).w = (_w), (_rect).h = (_h))
```

4.Library access functions

TwGfxOpen

Purpose	Access the accelerated graphics library. You can call this function as many times as desired. However, you must pair each `TwGfxOpen` with a `TwGfxClose` to avoid resource leaks.	
Prototype	`Err TwGfxOpen(TwGfxType** aResult, TwGfxInfoType* aInfoResult)`	
Parameters	`[out] aResult`	Pointer to a handle to the graphics library. If the request succeeds then `*aResult` is set to a handle to the graphics library for use in subsequent calls.
	`[inout] aInfoResult`	Pointer to a `TwGfxInfoType` object which is filled with a description of the capabilities of the device and the graphics library. Note that you must set the size field of `TwGfxInfoType` to the size of the data structure. This argument can be NULL if no such data is desired.
Result	`errNone` – Succeeded `TwGfxErrorLibraryOpen` – the library is already open `twGfxErrorBadObjectVersion` – the `TwGfxInfoType` size field doesn't match a known version for the library	
Header	`TwGfx.h`	

TwGfxClose

Purpose	Shutdown use of the library, releasing all resources associated with the library.	
Prototype	`Err TwGfxClose(TwGfxType* aGfx)`	
Parameters	`[in] aGfx`	A handle to the graphics library.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the library was invalid	
Side Effects	This function releases all resources allocated by the library including any surface objects that were created and not yet released. Note that only the surfaces created using this instance of the TwGfx library are released. Continuing to use the `aGfx` handle after calling close yields undefined results (most likely a crash).	
Header	`TwGfx.h`	

TwGfxGetInfo

Purpose	Query the graphics library for information describing the capabilities of the device and of the graphics library.	
Prototype	`Err TwGfxGetInfo(TwGfxType* aGfx, TwGfxInfoType* aInfoResult)`	
Parameters	`[in] aGfx`	A handle to the graphics library.
	`[inout] aInfoResult`	Pointer to a `TwGfxInfoType` object which will be filled in with a description of the capabilities of the device and of the graphics library. The size field of `TwGfxInfoType` must be set to the size of the data structure.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the library is invalid `twGfxErrorNullPointer` – the `aInfoResult` pointer is NULL `twGfxErrorBadObjectVersion` – the `TwGfxInfoType` size field doesn't match a known version for the library	
Header	`TwGfx.h`	

TwGfxGetMemoryUsage

Purpose	Query the current usage of graphics accelerator memory.	

Prototype	`Err TwGfxGetMemoryUsage(TwGfxType* aGfx,`
	`Int32 aLocation,`
	`Int32* aUsedResult)`	

Parameters	`[in] aGfx`	A handle to the graphics library.
	`[in] aLocation`	Must be `twGfxLocationAcceleratorMemory` or `twGfxLocationAcceleratorMemoryNoBackingStore`
	`[out] aUsedResult`	Pointer to an integer that receives the usage value.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the library is invalid `twGfxErrorNullPointer` – the `aUsedResult` pointer is NULL twGfxErrorInvalidLocation – the `aLocation` value is invalid	
Comments	This call returns the amount of accelerator memory **used** not the amount free as is returned in the TwGfxInfoType object's freeAcceleratorMemory field.	
Header	`TwGfx.h`	

TwGfxGetPalmDisplaySurface

Purpose	Query the graphics library for the surface used for the Palm OS display. The function returns the `TwGfxSurfaceType` handle that represents the actual display.	
Prototype	`Err TwGfxGetPalmDisplaySurface(TwGfxType* aGfx, TwGfxSurfaceType** aResult)`	
Parameters	`[in] aGfx`	A handle to the graphics library.
	`[out] aResult`	Pointer to a handle to a graphics surface that is filled in when the request succeeds.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the library is invalid `twGfxErrorNullPointer` – the `aResult` pointer is NULL	
Comments	The Palm display surface represents the subset of the display surface used by the Palm OS. It stays consistent with the size, shape, location, and orientation of the Palm OS display, including the "back buffer" used by `WinScreenLock`. Each instance of a library has its own unique reference to the Palm display surface. This means that changing the surface clip for the Palm display surface in one library handle will not affect it in a Palm display surface created from a different library handle. For example, if you open the TwGfx library twice, query the Palm display surface for each library instance, and then compare the two surface handles, ityields different values for the surface handles. Use this function – Don't use `TwGfxGetDisplaySurface`.	
Header	`TwGfx.h`	

TwGfxInVBlank

Purpose	Query the graphics library and see whether or not the display is in the vertical blanking period.	
Prototype	`Err TwGfxInVBlank(TwGfxType* aGfx, Boolean* aInVBlankResult)`	
Parameters	`[in] aGfx`	A handle to the graphics library.
	`[out] aInVBlankResult`	Pointer to a Boolean value which indicates the current state of the vertical blanking. The value is *true* if the display is in vertical blanking, *false* otherwise.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the library is invalid `twGfxErrorNullPointer` – the `aInfoResult` pointer is NULL	
Comments	See `TwGfxAsyncBlt` for a more efficient way to sync up with vertical blanking.	
Header	`TwGfx.h`	

TwGfxWaitForVBlank

Purpose	Wait for the vertical blanking period to begin. This function may return immediately if the display is already in the vertical blanking period.	
Prototype	`Err TwGfxWaitForVBlank(TwGfxType* aGfx)`	
Parameters	`[in] aGfx`	A handle to the graphics library.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the library is invalid	
Comments	See `TwGfxAsyncBlt` for a more efficient way to sync up with vertical blanking.	
Header	`TwGfx.h`	

TwGfxGetDisplaySurface

Purpose	Query the graphics library for the entire display surface and return the `TwGfxSurfaceType` handle that represents the actual display.	
Prototype	`Err TwGfxGetDisplaySurface(TwGfxType* aGfx, TwGfxSurfaceType** aResult)`	
Parameters	`[in] aGfx`	A handle to the graphics library.
	`[out] aResult`	Pointer to a handle to a graphics surface that is filled in when the request succeeds.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the library is invalid `twGfxErrorNullPointer` – the `aResult` pointer is NULL	
Comments	The display surface represents the entire display surface including the Pen Input Area and Status Area. Use `TwGfxGetPalmDisplaySurface` instead. It is nearly identical in function to this API, and has the added benefit of keeping in sync with the API's that affect the size, shape, location, and orientation of the Palm display surface. If you must use this function, make sure to close the Pen Input Area (PINS) using the `PINSetInputAreaState` API and the status area using the `StatHide`. These functions are defined in `PenInputMgr.h`, and if you lose focus, make sure to re-close Pen Input Area and Status Area before you begin drawing again. Each instance of a library has its own unique reference to the display surface. This means that changing the surface clip for the display surface in one library handle does not affect it in a display surface created from a different library handle. For example, if you open the TwGfx library twice, query the display surface for each library instance, and then compare the two surface handles, it yields different values for the surface handles.	

Header	`TwGfx.h`

5.Surface functions

TwGfxAllocSurface

Purpose	Attempt to allocate a new surface. Surfaces are used by the rendering functions described below. Surfaces should be allocated in accelerator memory.	
Prototype	`Err TwGfxAllocSurface(TwGfxType* aGfx, TwGfxSurfaceType** aResult, TwGfxSurfaceInfoType* aDescription);`	
Parameters	`[in] aGfx`	A handle to the graphics library.
	`[out] aResult`	Pointer to a handle to a surface that is filled in if the request succeeds.
	`[inout] aDescription`	Pointer to a `TwGfxSurfaceInfoType` object. This pointer must not be NULL and the size field must be initialized to the size of the data structure. In addition, you must set the width, height, pixelFormat and location fields to the values desired for creation of the surface. All other fields are ignored.

Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the library is invalid `twGfxErrorOutOfMemory` – the library ran out of memory `twGfxErrorInvalidPixelFormat` – the pixelFormat field is invalid `twGfxErrorInvalidLocation` – the location field is invalid `twGfxErrorInvalidSize` – the width/height fields are <= zero, or they are too large. `twGfxErrorNullPointer` – the `aDescription` pointer is NULL `twGfxErrorBadObjectVersion` – the `TwGfxSurfaceInfoType` size field doesn't match a known version for the library `twGfxErrorSurfaceAllocFailed` – the allocation failed (insufficient device memory is the most likely reason).
Side Effects	When the request succeeds, the fields in `aDescription` are filled in so that the requester receives a complete description of the surface.
Comments	Only the pixel format `twGfxPixelFormatRGB565_LE` can be used for surface creation at this time. There are currently two locations supported: `twGfxLocationAcceleratorMemory` and `twGfxLocationAcceleratorMemoryNoBackingStore`. The former allocates dynamic heap memory to maintain a copy of the surface during system sleep. The latter does not. The surface width or height is limited to a maximum of 8191 pixels.
Header	`TwGfx.h`

TwGfxFreeSurface

Purpose	Free a previously allocated surface.	
Prototype	`Err TwGfxFreeSurface(TwGfxType* aGfx, TwGfxSurfaceType* aSurface)`	
Parameters	`[in] aGfx`	A handle to the graphics library.
	`[in] aSurface`	Handle to the surface that should be freed.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the library or surface is invalid	
Comments	If you free the display surface or the Palm surface and then call `TwGfxGetDisplaySurface` or `TwGfxGetPalmDisplaySurface` (respectively) the surface object will be automatically recreated. Note that freeing the Palm or display surface will not free the video memory associated with those surfaces.	
Header	`TwGfx.h`	

TwGfxSetClip

Purpose	Set the clipping rectangle associated with the given surface. All rendering requests are clipped by the surface clipping rectangle.	
Prototype	`Err TwGfxSetClip(TwGfxSurfaceType* aSurface, const TwGfxRectType* aClipRect)`	
Parameters	`[in] aSurface`	A handle to the surface.
	`[in] aClipRect`	Pointer to the clipping rectangle. If `aClipRect` is NULL then clipping is disabled for the surface (this has the same effect as setting the clipping rectangle to cover the entire surface).
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the surface is invalid	
Comments	The clipping rectangle for all surfaces automatically intersects with the bounds of the actual clipping surface before it is applied. This is important when considering the Palm display surface because its bounds change dynamically. The clipping rectangle provided by this API remains in effect, but it intersects with the actual bounds of the Palm surface as it changes size, position and orientation.	
Header	`TwGfx.h`	

TwGfxGetClip

Purpose	Get the clipping rectangle associated with a surface.	
Prototype	`Err TwGfxGetClip(TwGfxSurfaceType* aSurface, TwGfxRectType* aResult)`	
Parameters	`[in] aSurface`	A handle to the surface.
	`[out] aClipRect`	Pointer to the rectangle object which will be filled in with the surface clipping rectangle.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the surface is invalid or the library is not open `twGfxErrorNullPointer` – the `aClipRect` pointer is NULL	
Comments	If the clipping for a surface is disabled (the default state for a surface) then the fields in `aClipRect` will contain the size of the surface (0, 0, width, height).	
Header	`TwGfx.h`	

TwGfxGetSurfaceInfo

Purpose	Query the surface information for a given surface.	
Prototype	`Err TwGfxGetSurfaceInfo(TwGfxSurfaceType* aSurface, TwGfxSurfaceInfoType* aResult)`	
Parameters	`[in] aSurface`	A handle to the surface.
	`[inout] aResult`	A `TwGfxSurfaceInfoType` object that will be filled in with the description of the surface. The size field of `aResult` must be set to the size of the data structure before calling this function.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the surface is invalid or the library is not open `twGfxErrorNullPointer` – the `aClipRect` pointer is NULL `twGfxErrorBadObjectVersion` – the `TwGfxSurfaceInfoType` size field doesn't match a known version for the library	
Header	`TwGfx.h`	

TwGfxLockSurface

Purpose	Return a memory address for the surfaces video memory.	
Prototype	`Err TwGfxLockSurface(TwGfxSurfaceType* aSurface, void** aAddressResult)`	
Parameters	`[in] aSurface`	A handle to the surface.
	`[out] aAddressResult`	A pointer to a void pointer that will be filled in with a readable/writable memory address for the surface memory.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the surface is invalid or the library is not open `twGfxErrorNullPointer` – the `aAddressResult` pointer is NULL	
Comments	The memory address returned remains valid until `TwGfxUnlockSurface` is called. Note that you can nest these functions. When this occurs, the same memory address is always returned and is valid until the matching number of `TwGfxUnlockSurface` requests are made. Use care when mixing direct access to the surface memory with surface rendering operations. Surface rendering operations can execute in parallel with the CPU's access to the memory and yield unpredictable results. To work around this issue, use `TwGfxUnlockSurface` to release access to the surface followed by `TwGfxLockSurface`. `TwGfxLockSurface` ensures that the rendering pipeline is empty before returning a memory address.	
Header	`TwGfx.h`	

TwGfxUnlockSurface

Purpose	Unlock a previously locked surface.	
Prototype	`Err TwGfxUnlockSurface(TwGfxSurfaceType* aSurface, Boolean aUpdate)`	
Parameters	`[in] aSurface`	A handle to the surface.
	`[in] aUpdate`	This flag indicates to the graphics library that the surface memory was modified by the requester. If for some reason actual surface memory was not returned by `TwGfxLockSurface`, it should be copied to the actual surface memory.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the surface is invalid or the library is not open `twGfxErrorSurfaceNotLocked` – the surface was not locked	
Header	`TwGfx.h`	

TwGfxReadSurface

Purpose	Make a copy of the surface display memory.	
Prototype	`Err TwGfxReadSurface(TwGfxSurfaceType* aSurface, void* aDest, UInt8 aAsync)`	
Parameters	`[in] aSurface`	A handle to the surface.
	`[out] aDest`	A pointer to the memory where the surface memory will be written. It is the requester's responsibility to allocate enough memory.
	`[in] aAsync`	When set to twGfxTransferAsync this flag indicates that the copy should be done asynchronously. Use `TwGfxSurfaceIsReady` to query when the copy is done.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the surface is invalid or the library is not open `twGfxErrorNullPointer` – the `aDest` pointer is NULL `twGfxErrorOperationInProgress` – a `TwGfxSurfaceRead` or `TwGfxSurfaceWrite` operation is already in progress	
Comments	This is identical to calling `TwGfxReadSurfaceRegion` with a bounding rectangle that covers the entire surface. To allocate enough memory for the copy, use `TwGfxGetSurfaceInfo` and perform this calculation: `Int32 bytesNeeded = info.rowBytes * info.height;`	
Header	`TwGfx.h`	

TwGfxReadSurfaceRegion

Purpose	Make a copy of the surface display memory.	
Prototype	`Err TwGfxReadSurfaceRegion(TwGfxSurfaceType* aSurface, const TwGfxRectType* aBounds, void* aDestPixels, Int32 aDestRowBytes, UInt8 aAsync)`	
Parameters	`[in] aSurface`	A handle to the surface.
	`[in] aBounds`	A pointer to the rectangle object which contains the area of the surface to be written.
	`[out] aDestPixels`	A pointer to the memory where the surface memory will be written. It is the requester's responsibility to allocate enough memory.
	`[in] aDestRowBytes`	The number of data bytes per row in `aDestPixels`.
	`[in] aAsync`	When set to twGfxTransferAsync this flag indicates that the copy should be asynchronous. Use `TwGfxSurfaceIsReady` to query when the copy is done.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the surface is invalid or the library is not open `twGfxErrorNullPointer` – the aDest pointer is NULL `twGfxErrorOperationInProgress` – an asynchronous operation is in progress	
Header	`TwGfx.h`	

TwGfxWriteSurface

Purpose	Write a memory buffer to the surface display memory.	
Prototype	`Err TwGfxWriteSurface(TwGfxSurfaceType* aSurface, const void* aSource, UInt8 aAsync)`	
Parameters	`[in] aSurface`	A handle to the surface.
	`[in] aSource`	A pointer to the memory to be copied to the surface display memory.
	`[in] aAsync`	When set to twGfxTransferAsync this flag indicates that the copy should be asynchronous. Use `TwGfxIsSurfaceReady` to query when the copy is done.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the surface is invalid or the library is not open `twGfxErrorNullPointer` – the `aSource` pointer is NULL `twGfxErrorOperationInProgress` – an asynchronous operation is already in progress	
Comments	See TwGfxReadSurface for an example of how to allocate enough memory for the copy. This function is the same as calling `TwGfxWriteSurfaceRegion` and specifying a region that covers the entire surface with source row bytes that are the same as the surface row bytes.	
Header	`TwGfx.h`	

TwGfxWriteSurfaceRegion

Purpose	Write a memory buffer to a subset of the surface display memory.	
Prototype	`Err TwGfxWriteSurfaceRegion(TwGfxSurfaceType* aSurface, const TwGfxRectType* aBounds, const void* aSourcePixels, Int32 aSourceRowBytes, Boolean aAsync)`	
Parameters	`[in] aSurface`	A handle to the surface.
	`[in] aBounds`	A pointer to the rectangle object which contains the area of the surface to be written.
	`[in] aSourcePixels`	A pointer to the memory to be copied to the surface display memory.
	`[in] aSourceRowBytes`	The number of bytes per row of data in `aSourcePixels`.
	`[in] aAsync`	When set to twGfxTransferAsync this flag indicates that the copy should be asynchronous. Use `TwGfxIsSurfaceReady` to query when the copy is done.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the surface is invalid or the library is not open `twGfxErrorNullPointer` – the `aSource` pointer is NULL `twGfxErrorInvalidCoord` – the coordinates in `aDestRect` are outside the bounds of the surface or specify an empty area (zero width/height) `twGfxErrorOperationInProgress` – a `TwGfxSurfaceRead` or `TwGfxSurfaceWrite` operation is already in progress	

Comments	The amount of memory necessary for the copy to work properly depends on the pixel format of the surface and the height of the `aDestRect` and `aSourceRowBytes`. For example, assuming 2 bytes per pixel, the following calculation will be correct: `totalBytes = aDestRect->h * aSourceRowBytes * 2;`
Header	`TwGfx.h`

TwGfxIsSurfaceReady

Purpose	Query the surface and see if it is ready for another TwGfxSurfaceRead or TwGfxSurfaceWrite request.	
Prototype	`Err TwGfxIsSurfaceReady(TwGfxSurfaceType* aSurface)`	
Parameters	`[in] aSurface`	A handle to the surface.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the surface is invalid or the library is not open `twGfxErrorOperationInProgress` – a `TwGfxSurfaceRead` or `TwGfxSurfaceWrite` operation is still in progress	
Comments	This function returns `errNone` when no operations are pending (and the surface is a valid surface). If the arguments are valid and an operation is pending, then it returns `twGfxErrorOperationInProgress`.	
Header	`TwGfx.h`	

6.Rendering functions

This section describes the rendering methods in the graphics library.

TwGfxBitBlt

Purpose	Call a basic bitblt rendering function.	
Prototype	`Err TwGfxBitblt(TwGfxSurfaceType* aDestSurface, const TwGfxPointType* aDestPoint, TwGfxSurfaceType* aSourceSurface, const TwGfxRectType* aSourceRect)`	
Parameters	`[in] aDestSurface`	A handle to the surface to which you want to bitblt.
	`[in] aDestPoint`	The upper-left coordinate in the destination surface to which you want to bitblt.
	`[in] aSourceSurface`	A handle to the surface from which you want to bitblt.
	`[in] aSourceRect`	The area in the source surface from which you want to bitblt.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – aDestPoint or aSourceRect is NULL `twGfxErrorInvalidCoord` – the coordinate values in the destination point or the source rectangle address values outside their respective surfaces `twGfxErrorOperationInProgress` – an asynchronous operation is in progress	
Header	`TwGfx.h`	

TwGfxAsyncBlt

Purpose	Call a basic bitblt rendering function, done asynchronously.	
Prototype	`Err TwGfxAsyncblt(TwGfxSurfaceType* aDestSurface, const TwGfxPointType* aDestPoint, TwGfxSurfaceType* aSourceSurface, const TwGfxRectType* aSourceRect)`	
Parameters	`[in] aDestSurface`	A handle to the surface to which you want to bitblt.
	`[in] aDestPoint`	The upper-left coordinate in the destination surface to which you want to bitblt.
	`[in] aSourceSurface`	A handle to the surface from which you want to bitblt.
	`[in] aSourceRect`	The area in the source surface from which you want to bitblt.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – `aDestPoint` or `aSourceRect` is NULL `twGfxErrorInvalidCoord` – the coordinate values in the destination point or the source rectangle address values outside their respective surfaces `twGfxErrorOperationInProgress` – an asynchronous operation is in progress or another asynchronous bitblt has been requested	
Comments	This function is identical to `TwGfxBitBlt` except that the bitblt occurs during the next vertical retrace period. There can only one asynchronous bitblt request at a time.	
Header	`TwGfx.h`	

TwGfxTransparentBlt

Purpose	Call a transparent bitblt rendering function.	
Prototype	`Err TwGfxTransparentBlt(TwGfxSurfaceType* aDestSurface, const TwGfxPointType* aDestPoint, TwGfxSurfaceType* aSourceSurface, const TwGfxRectType* aSourceRect, TwGfxPackedRGBType aTransparentColor)`	
Parameters	`[in] aDestSurface`	A handle to the surface to which you want to bitblt.
	`[in] aDestPoint`	The upper-left coordinate in the destination surface to which you want to bitblt.
	`[in] aSourceSurface`	A handle to the surface from which you want to bitblt.
	`[in] aSourceRect`	The area in the source surface from which you want to bitblt.
	`[in] aTransparentColor`	The color which should not be rendered.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – `aDestPoint` or `aSourceRect` is NULL `twGfxErrorInvalidCoord` – the coordinate values in the destination point or the source rectangle address values outside their respective surfaces `twGfxErrorOperationInProgress` – an asynchronous operation is in progress	
Comments	The source surface is bitblt'd to the destination surface like a regular bitblt with one distinction: any pixel in the source surface whose color is identical to `aTransparentColor` is not written to the destination surface. Since `aTransparentColor` may have higher resolution than the actual display	

	surface you may wish to use `TwGfxMakeDisplayRGB` and then convert that to packed format with `TwGfxDisplayRGBToPackedRGB`.
Header	`TwGfx.h`

TwGfxMaskBlt

Purpose	Call a mask bitblt rendering function.	
Prototype	`Err TwGfxMaskBlt(TwGfxSurfaceType* aDestSurface, const TwGfxPointType* aDestPoint, TwGfxSurfaceType* aSourceSurface, const TwGfxRectType* aSourceRect, const TwGfxBitmapType* aMask)`	
Parameters	`[in] aDestSurface`	A handle to the surface to which you want to bitblt.
	`[in] aDestPoint`	The upper-left coordinate in the destination surface to which you want to bitblt.
	`[in] aSourceSurface`	A handle to the surface from which you want to bitblt.
	`[in] aSourceRect`	The area in the source surface from which you want to bitblt.
	`[in] aMask`	A monochrome bitmap that describes which pixels to render and which to not render. The width and the height of the mask must be >= the width and height of the source rect.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – `aDestPoint` or `aSourceRect` is NULL `twGfxErrorInvalidCoord` – the coordinate values in the destination point or the source rectangle address values outside their respective surfaces `twGfxErrorOperationInProgress` – an asynchronous operation is in progress `twGfxErrorInvalidSize` – the mask bitmap width/height/rowbytes are too	

	small.
Comments	Only monochrome bitmaps can be used as masks. When a one bit is present in the bitmap the source surface pixel will be written to the destination surface. When a zero bit is present the destination pixel will remain unchanged. The mask data rowbytes must be a multiple of sizeof(UInt32) and the mask data must be aligned on a UInt32 boundary.
Header	TwGfx.h

TwGfxBlendBlt

Purpose	Call a blending bitblt rendering function.	
Prototype	`Err TwGfxBlendBlt(TwGfxSurfaceType* aDestSurface, const TwGfxPointType* aDestPoint, TwGfxSurfaceType* aSourceSurface, const TwGfxRectType* aSourceRect, TwGfxPackedRGBType aSourceAlpha)`	
Parameters	`[in] aDestSurface`	A handle to the surface to which you want to bitblt.
	`[in] aDestPoint`	The upper-left coordinate in the destination surface to which you want bitblt.
	`[in] aSourceSurface`	A handle to the surface from which you want to bitblt.
	`[in] aSourceRect`	The area in the source surface from which you want to bitblt.
	`[in] aSourceAlpha`	The constant alpha color to use over the entire surface
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – `aDestPoint` or `aSourceRect` is NULL `twGfxErrorInvalidCoord` – the coordinate values in the destination point or the source rectangle address values outside their respective surfaces `twGfxErrorOperationInProgress` – an asynchronous operation is in progress	

Comments	The source surface is blended into the destination surface using the following formula on a per pixel basis:
	
	`dst.r = src.r * alpha.r + (1 – alpha.r) * dst.r`
	`dst.g = src.g * alpha.g + (1 – alpha.g) * dst.g`
	`dst.b = src.b * alpha.b + (1 – alpha.b) * dst.b`
	
	The alpha color componets are logically normalized to a 0.0 – 1.0 range (inclusive) before the above calculation is done.
	
	The above formula can be translated to "As the alpha value increases towards 1, more of the source pixel is used and less of the destination pixel is used."

Header	`TwGfx.h`

TwGfxMaskBlendBlt

Purpose	Call a blending bitblt rendering function.	
Prototype	`Err TwGfxMaskBlendBlt(TwGfxSurfaceType* aDestSurface, const TwGfxPointType* aDestPoint, TwGfxSurfaceType* aSourceSurface, const TwGfxRectType* aSourceRect, const TwGfxBitmapType* aAlphaMask)`	
Parameters	`[in] aDestSurface`	A handle to the surface to which you want bitblt.
	`[in] aDestPoint`	The upper-left coordinate in the destination surface to which you want bitblt.
	`[in] aSourceSurface`	A handle to the surface from which you want to bitblt.
	`[in] aSourceRect`	The area in the source surface from which you want to bitblt.
	`[in] aAlphaMask`	A bitmap containing per-pixel alpha values. The width and the height of the mask must be >= the width and height of the source rect.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – `aDestPoint` or `aSourceRect` is NULL `twGfxErrorInvalidCoord` – the coordinate values in the destination point or the source rectangle address values outside their respective surfaces `twGfxErrorOperationInProgress` – an asynchronous operation is in progress `twGfxErrorInvalidSize` – the alpha mask bitmap width/height/rowbytes are too small.	

Comments	The alpha mask bitmap provides a per-pixel alpha value to perform the blending calculation with (see TwGfxBlendBlt for the blending calculation). The alpha mask pixel format must be one of the index formats (monochrome, 2bpp, 4bpp or 8bpp) with the 4bpp being the most efficient choice **(and the only supported format at this time)**. Each pixel of the source surface in the source rectangle is blended with the destination pixel using the blending formula described in `TwGfxBlendBlt`. The alpha color value is taken from the mask, which means that each pixel can have a different alpha value applied to the calculation.
Header	`TwGfx.h`

TwGfxStretchBlt

Purpose	Call a stretching bitblt rendering function.	

Prototype	`Err TwGfxStretchBlt(TwGfxSurfaceType* aDestSurface, const TwGfxRectType* aDestRect, TwGfxSurfaceType* aSourceSurface, const TwGfxRectType* aSourceRect)`
	
	`Err TwGfxStretchBlt2(TwGfxSurfaceType* aDestSurface, const TwGfxRectType* aDestRect, TwGfxSurfaceType* aSourceSurface, const TwGfxRectType* aSourceRect, UInt32 aStretchFlags)`	

Parameters	`[in] aDestSurface`	A handle to the surface to which you want bitblt.
	`[in] aDestRect`	The area in the destination surface to which you want bitblt.
	`[in] aSourceSurface`	A handle to the surface from which you want to bitblt.
	`[in] aSourceRect`	The area in the source surface from which you want to bitblt.
	`[in] aStretchFlags`	Flags specifying how the stretch blt should be done. The following flag settings are available: twGfxStretchFast means use the fatest possible approach which may yield an inexact match to the aDestRect area. Note that performance will vary the most with this setting depending upon whether or not the graphics acclerator ends up doing the stretch blt or not (this decision depends upon the relationship between the source and destination dimensions). twGfxStretchExact means use an exact stretch that will match exactly the aDestRect area. This setting will not use the

		graphics accelerator directly; however, performance will be consistent regardless of the source and destination dimensions. twGfxStretchSmooth means apply smoothing instead of pixel replication. This flag can be combined with twGfxStretchFast or twGfxStretchExact.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – `aDestPoint` or `aSourceRect` is NULL `twGfxErrorInvalidCoord` – the coordinate values in the destination point or the source rectangle address values outside their respective surfaces `twGfxErrorOperationInProgress` – an asynchronous operation is in progress `twGfxErrorInvalidFlags` – the flags specified by aStretchFlags are invalid	
Comments	The source surface is *stretched* or *shrunk* to fit in the destination rectangle. The exact effect on the pixels is not defined here. The TwGfxStretchBlt operation is not fast and is only marginally accelerated by the graphics accelerator. TwGfxStretchBlt is equivalent in function to calling TwGfxStretchBlt2 with the aStretchFlags argument set to "twGfxStretchSmooth | twGfxStretchExact". For TwGfxStretchBlt2 the aStretchFlags argument is specified by the caller and gives the caller full control of the outcome. Regardless of which flag settings your application uses, please test and see if it meets your needs. Note that the twGfxStretchExact flag is the highest precedence - if it's set then the graphics accelerator will not be directly used to do the stretch blt.	
Header	`TwGfx.h`	

TwGfxTileBlt

Purpose	Call a tiling bitblt rendering function.	
Prototype	`Err TwGfxTileBlt(TwGfxSurfaceType* aDestSurface, const TwGfxRectType* aDestRect, TwGfxSurfaceType* aSourceSurface, const TwGfxPointType* aSourceAlignmentPoint)`	
Parameters	`[in] aDestSurface`	A handle to the surface to which you want bitblt.
	`[in] aDestRect`	The destination area to tile with the source surface.
	`[in] aSourceSurface`	A handle to the surface from which you want to bitblt.
	`[in] aSourceAlignmentPoint`	The offset in x & y from which to begin the tiling.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – `aDestPoint` or `aSourceRect` is NULL `twGfxErrorInvalidCoord` – the coordinate values in the destination point or the source rectangle address values outside their respective surfaces `twGfxErrorOperationInProgress` – an asynchronous operation is in progress	
Comments	The source surface is drawn as many times as necessary to cover the destination rectangle. The first pixel written to the upper-left corner of the destination rectangle will come from the source surface from the `aSourceAlignmentPoint` (x,y) offset.	
Header	`TwGfx.h`	

TwGfxTransformBlt

Purpose	Call a transforming bitblt rendering function.	
Prototype	`Err TwGfxTransformBlt(TwGfxSurfaceType* aDestSurface, const TwGfxPointType* aDestPoint, TwGfxSurfaceType* aSourceSurface, const TwGfxRectType* aSourceRect, Int32 aRotationFlags, Int32 aMirrorFlags)`	
Parameters	`[in] aDestSurface`	A handle to the surface to which you want bitblt.
	`[in] aDestPoint`	The upper-left coordinate in the destination surface to which you want bitblt.
	`[in] aSourceSurface`	A handle to the surface from which you want to bitblt.
	`[in] aSourceRect`	The area in the source surface from which you want to bitblt.
	`[in] aRotationFlags`	A value indicating the kind of rotation to perform.
	`[in] aMirrorFlags`	A value indicating the kind of mirroring to perform.

Result	errNone – Succeeded twGfxErrorInvalidHandle – the handle to a surface is invalid or the library is not open twGfxErrorNullPointer – aDestPoint or aSourceRect is NULL twGfxErrorInvalidCoord – the coordinate values in the destination point or the source rectangle address values are outside their respective surfaces twGfxErrorInvalidRotation – the rotation value is invalid twGfxErrorInvalidMirror – the mirror value is invalid twGfxErrorOperationInProgress – an asynchronous operation is in progress
Comments	Legal values for aRotationFlags: twGfxRotateNone, twGfxRotateCW90, twGfxRotateCW180, twGfxRotateCW270, twGfxRotateCCW90, twGfxRotateCCW180, twGfxRotateCCW270 Legal values for aMirrorFlags: twGfxMirrorNone, twGfxMirrorHorizontal, twGfxMirrorVertical, twGfxMirrorBoth
Header	TwGfx.h

TwGfxDrawPoints

Purpose	Draw a set of points to the destination surface.	
Prototype	`Err TwGfxDrawPoints(TwGfxSurfaceType* aDestSurface, const TwGfxPointType* aPoints, Int32 aNumberOfPoints, TwGfxPackedRGBType aColor)`	
Parameters	`[in] aDestSurface`	A handle to the surface.
	`[in] aPoints`	A pointer to the array of `TwGfxPointType` objects containing the x,y coordinates of the points to be plotted.
	`[in] aNumberOfPoints`	The number of points to plot.
	`[in] aColor`	The color to use when plotting each point.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to the surface is invalid or the library is not open `twGfxErrorNullPointer` – `aPoints` is NULL `twGfxErrorInvalidCount` – the `aNumberOfPoints` value is <= zero `twGfxErrorOperationInProgress` – an asynchronous operation is in progress	
Header	`TwGfx.h`	

TwGfxDrawColorPoints

Purpose	Draw a set of points to the destination surface. Each point has its own color.	
Prototype	`Err TwGfxDrawColorPoints(TwGfxSurfaceType* aDestSurface, const TwGfxPointType* aPoints, Int32 aNumberOfPoints, const TwGfxPackedRGBType* aColors)`	
Parameters	`[in] aDestSurface`	A handle to the surface.
	`[in] aPoints`	A pointer to the array of `TwGfxPointType` objects containing the x,y coordinates of the points to be plotted.
	`[in] aNumberOfPoints`	The number of points to plot.
	`[in] aColors`	The color to use when plotting each point. There must be one color in the `aColors` array for each point in the `aPoints` array.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – `aPoints` or `aColors` is NULL `twGfxErrorInvalidCount` – the `aNumberOfPoints` value is <= zero `twGfxErrorOperationInProgress` – an asynchronous operation is in progress	
Header	`TwGfx.h`	

TwGfxDrawLines

Purpose	Draw one or more connected lines.	
Prototype	`Err TwGfxDrawLines(TwGfxSurfaceType* aDestSurface, const TwGfxPointType* aPoints, Int32 aNumberOfPoints, TwGfxPackedRGBType aColor)`	
Parameters	`[in] aDestSurface`	A handle to the surface.
	`[in] aPoints`	The set of points that define the lines to be drawn. Lines are drawn starting at `(aPoints[I-1].x,aPoints[I-1].y)` to `(aPoints[I].x,aPoints[I].y)` where I goes from 1 to `aNumberOfPoints-1`. Therefore, if `aNumberOfPoints` is 3 then 2 lines will be drawn.
	`[in] aNumberOfPoints`	The number of points in `aPoints`.
	`[in] aColor`	The color to use when drawing each line.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – `aPoints` is NULL `twGfxErrorInvalidCount` – the number of points is less than two. `twGfxErrorOperationInProgress` – an asynchronous operation is in progress	

Comments	Example: If `aNumberOfPoints` is 3 then the following two lines are drawn:
	`(aPoints[0].x,aPoints[0].y) to (aPoints[1].x,aPoints[1].y)`
	`(aPoints[1].x,aPoints[1].y) to (aPoints[2].x,aPoints[2].y)`	

Header	`TwGfx.h`	

TwGfxDrawLineSegments

Purpose	Draw one or more independent lines.	
Prototype	`Err TwGfxDrawLineSegments(TwGfxSurfaceType* aDestSurface, const TwGfxPointType* aPoints, Int32 aNumberOfPoints, TwGfxPackedRGBType aColor)`	
Parameters	`[in] aDestSurface`	A handle to the surface.
	`[in] aPoints`	The set of points that define the lines to be drawn.
	`[in] aNumberOfPoints`	The number of points in `aPoints`.
	`[in] aColor`	The color to use when drawing each line.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – aPoints is NULL `twGfxErrorInvalidCount` – the number of points is not even or is <= zero `twGfxErrorOperationInProgress` – an asynchronous operation is in progress	

Comments	Example: If `aNumberOfPoints` is 4 then the following two lines are drawn:
	`(aPoints[0].x,aPoints[0].y) to (aPoints[1].x,aPoints[1].y)`
	`(aPoints[2].x,aPoints[2].y) to (aPoints[3].x,aPoints[3].y)`	

Header	`TwGfx.h`	

TwGfxDrawRect

Purpose	Draw the outline of a rectangle.	
Prototype	`Err TwGfxDrawRect(TwGfxSurfaceType* aDestSurface, const TwGfxRectType* aRect, TwGfxPackedRGBColor aColor)`	
Parameters	`[in] aDestSurface`	A handle to the surface.
	`[in] aRect`	A pointer to the rectangle to outline.
	`[in] aColor`	The color to use when drawing the outline.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – aRect is NULL `twGfxErrorOperationInProgress` – an asynchronous operation is in progress	
Header	`TwGfx.h`	

TwGfxFillRect

Purpose	Fill a rectangle with the given color.	
Prototype	`Err TwGfxFillRect(TwGfxSurfaceType* aDestSurface, const TwGfxRectType* aRect, TwGfxPackedRGBColor aColor)`	
Parameters	`[in] aDestSurface`	A handle to the surface.
	`[in] aRect`	A pointer to the rectangle to fill.
	`[in] aColor`	The color to use when filling the rectangle.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – `aRect` is NULL `twGfxErrorOperationInProgress` – an asynchronous operation is in progress	
Header	`TwGfx.h`	

TwGfxDrawSpans

Purpose	Draw one or more horizontal spans with a constant color.	
Prototype	`Err TwGfxDrawSpans(TwGfxSurfaceType* aDestSurface, const TwGfxSpanType* aSpans, Int32 aNumberOfSpans, TwGfxPackedRGBColor aColor)`	
Parameters	`[in] aDestSurface`	A handle to the surface.
	`[in] aSpans`	A pointer to one or more `TwGfxSpanType` objects.
	`[in] aNumberOfSpans`	The number of spans to draw.
	`[in] aColor`	The color to draw each span.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – `aSpans` is NULL `twGfxErrorInvalidCount` – the number of spans is <= zero `twGfxErrorOperationInProgress` – an asynchronous operation is in progress	
Header	`TwGfx.h`	

TwGfxDrawBitmap

Purpose	Draw a bitmap to the destination surface.	
Prototype	`Err TwGfxDrawBitmap(TwGfxSurfaceType* aDestSurface, const TwGfxPointType* aDestPoint, const TwGfxBitmapType* aBitmap)`	
Parameters	`[in] aDestSurface`	A handle to the surface.
	`[in] aDestPoint`	The destination coordinates in the destination surface (upper-left) to draw the bitmap.
	`[in] aBitmap`	A pointer to the `TwGfxBitmapType` object which describes the size and pixel format of the bitmap. Note that the size field must be set to the size of the `TwGfxBitmapType` data structure.
Result	`errNone` – Succeeded `twGfxErrorInvalidHandle` – the handle to a surface is invalid or the library is not open `twGfxErrorNullPointer` – `aDestPoint` or `aBitmap` is NULL `twGfxErrorBadObjectVersion` – the `TwGfxBitmapType` size field doesn't match a known version for the library `twGfxErrorOperationInProgress` – an asynchronous operation is in progress `twGfxErrorInvalidSize` – the bitmap rowbytes value is too small.	

Comments	The following pixel formats are supported for bitmaps:
	
	`twGfxPixelFormatMonochrome`
	`twGfxPixelFormat2bpp`
	`twGfxPixelFormat4bpp`
	`twGfxPixelFormat8bpp`
	`twGfxPixelFormatRGB565_LE`
	`twGfxPixelFormatRGB565_BE`
	
	For index formats the palette field in `aBitmap` must point to a table of color values used to convert the index data into display pixels.
	
	The bitmap rowbytes must be a multiple of sizeof(UInt32) for the monochrome pixel format. For the 2bpp pixel format, the rowbytes must be >= (bitmap.width + 3) / 4. For the 4bpp pixel format, the rowbytes must be >= (bitmap.width + 1) / 2. For the 8bpp pixel format the rowbytes must be >= bitmap.width. For the 16bpp pixel formats the rowbytes must be >= 2 * bitmap.width.

Header	`TwGfx.h`