Introduction

Graphics Importer components provide a generic way for applications to interact with the images contained within any graphics file containing still images. The goal is to allow an application to work with any type of image data, regardless of the file format or compression used in the document. The application simply specifies the file that the image resides in, and the destination rectangle the image should be drawn into, and all other details will be taken care of. Of course, more complex interactions are supported.

The following routine shows the simplest possible use of graphics importers to draw a picture.

```
void drawFile(const FSSpec *fss, const Rect *boundsRect)
{
      GraphicsImportComponent gi;

      GetGraphicsImporterForFile(fss, &gi);

      GraphicsImportSetBoundsRect(gi, boundsRect);
      GraphicsImportDraw(gi);

      CloseComponent(gi);
}
```

This same code can be used to draw the file regardless of image file formats. In the current implementation the following file formats are supported: QuickDraw picture, MacPaint, Photoshop (2.5/3.0), SGI .rgb, GIF, and JFIF. Also supported is the new QuickTime Image File format, which is defined below.

If the same file is going to be drawn more than once, it is probably a good idea to keep the graphics import component around, rather than creating and disposing it each time. In this way, drawing will be faster as less work needs to be done for each draw.

Specifying the Data Source

Data for graphics import components can actually be stored in any location. Graphics import components use QuickTime data handler components to obtain their data. Most users of graphics import components will never need to worry about the details of this however, as routines are provided to allow the data source to be set as either a file or a handle.

```
ComponentResult GraphicsImportSetDataFile(GraphicsImportComponent
      gi, const FSSpec *theFile)
```

Specify the file that the graphics data resides in.

```
ComponentResult GraphicsImportGetDataFile(GraphicsImportComponent
      gi, FSSpec *theFile)
```

Returns the file that the graphics data resides in. If the data source is not a file, this routine returns paramErr.

```
    ComponentResult
          GraphicsImportSetDataHandle(GraphicsImportComponent gi,
          Handle h)
```
Specifies the handle that the graphics data resides in.

```
    ComponentResult
          GraphicsImportGetDataHandle(GraphicsImportComponent gi,
          Handle *h)
```
Returns the handle that the graphics data resides in. If the data source is not a handle, this routine returns paramErr.

```
    ComponentResult
          GraphicsImportSetDataReference(GraphicsImportComponent gi,
          Handle dataRef, OSType dataReType)
```
Specifies the data reference that the graphics data resides in. The routines GraphicsImportSetDataFile and GraphicsImportSetDataHandle both call this routine, with the appropriate data reference and data references type. See QuickTime documentation for more details on data references.

```
    ComponentResult
          GraphicsImportGetDataReference(GraphicsImportComponent gi,
          Handle *dataRef, OSType *dataRefType)
```
Returns the data reference that the graphics data resides in. Both the dataRef and dataRefType parameters may be set to nil if the corresponding information is not desired. Both GraphicsImportGetDataHandle and GraphicsImportGetDataFile call GraphicsImporGetDataReference and then manipulate the result accordingly.

Retrieving Image Data

```
    ComponentResult GraphicsImportReadData(GraphicsImportComponent gi,
          void *dataPtr, unsigned long dataOffset, unsigned long
          dataSize)
```
Once the data source has been assigned, data can be read directly from the image by using GraphicsImportReadData. GraphicsImportReadData communicates with the appropriate data handler to retrieve data. Typically only authors of graphics import component will need to use this routine. This routine should always be used to retrieve data from the data source, rather than reading it directly.

Getting Image Characteristics

```
    ComponentResult
          GraphicsImportGetNaturalBounds(GraphicsImportComponent gi,
          Rect *naturalBounds)
```
Use GraphicsImportGetNaturalBounds to determine the native size of the image associated with the graphics import component. The natural bounds will always be zero based. GraphcisImportGetNaturalBounds is provided for the convenience of users of graphics import components: the implementation just calls GraphicsImportGetImageDescription and extracts the width and height fields.

```
    ComponentResult
          GraphicsImportGetImageDescription(GraphicsImportComponent
          gi, ImageDescriptionHandle *desc)
```

GraphicsImportGetImageDescription returns lots of information about the image associated with the graphics import component. In particular, the format of the compressed data, bit depth, natural bounds, and resolution are returned. The returned image description must be disposed of by the caller.

Setting Drawing Parameters

When drawing, you may wish to specify various parameters for the drawing operation such as clipping, scaling, graphics mode, and quality, These are all supported by individual calls. All of these routines are based on corresponding routines in the Image Compression Manager for working with image decompression sequences.

```
      ComponentResult
            GraphicsImportSetBoundsRect(GraphicsImportComponent gi,
            const Rect *bounds)
```
GraphicsImportSetBoundsRect defines the rectangle into which the graphics image should be drawn. This routine creates a matrix to map from the image’s natural bounds to the specified bounds and calls GraphicsImportSetMatrix.

```
      ComponentResult
            GraphicsImportGetBoundsRect(GraphicsImportComponent gi, Rect
            *bounds)
```
GraphicsImportGetBoundsRect returns the bounding rectangle for drawing. This routine is implemented by calling GraphicsImportGetMatrix and GraphicsImportGetNaturalBounds and using the results to calculate the drawing rectangle.

```
      ComponentResult GraphicsImportSetMatrix(GraphicsImportComponent
            gi, const MatrixRecord *matrix)
```
Defines the matrix to be used for drawing.

```
      ComponentResult GraphicsImportGetMatrix(GraphicsImportComponent
            gi, MatrixRecord *matrix)
```
Returns the matrix to be used for drawing. The drawing matrix is initialized to the identity matrix when the graphics import component is instantiated.

```
      ComponentResult GraphicsImportSetClip(GraphicsImportComponent gi,
            RgnHandle clipRgn)
```
Defines the clipping region for drawing. All drawing operations ignore the port clip, so to clip the image, GraphicsImportSetClip must be called. Pass nil for the clipRgn to set the clipRgn to full. The default clip region is nil. The graphics import component makes a copy of this region.

```
      ComponentResult GraphicsImportGetClip(GraphicsImportComponent gi,
            RgnHandle *clipRgn)
```
Returns the current clipping region. The caller is responsible for disposing of the returned region. If there is no clipping region, nil is returned.

```
      ComponentResult
            GraphicsImportSetGraphicsMode(GraphicsImportComponent gi,
            long graphicsMode, const RGBColor *opColor)
```

Defines the graphics mode for drawing the image. If the opColor field is non-nil, the op color is also defined. The default graphics mode is ditherCopy and the default opColor is a 50% gray.

```
      ComponentResult
            GraphicsImportGetGraphicsMode(GraphicsImportComponent gi,
            long *graphicsMode, RGBColor *opColor)
```
Returns the current graphics mode and opColor. Both the graphicsMode and opColor fields may be set to nil if the corresponding information is not required.

```
      ComponentResult GraphicsImportSetQuality(GraphicsImportComponent
            gi, CodecQ quality)
```
Sets the image display quality. In the Image Compression Manager this is called accuracy. The default value is normal accuracy. Values range from 0 to 1024, with higher values being higher quality.

```
      ComponentResult GraphicsImportGetQuality(GraphicsImportComponent
            gi, CodecQ *quality)
```
Returns the image display quality. In the Image Compression Manager this is called accuracy.

```
      ComponentResult
            GraphicsImportSetSourceRect(GraphicsImportComponent gi,
            const Rect *sourceRect)
```
Defines the source rectangle area of the image to use. Provides a way to only use a portion of the source image. Pass nil to specifiy that the entire image source rectangle should be used.

```
      ComponentResult
            GraphicsImportGetSourceRect(GraphicsImportComponent gi, Rect
            *sourceRect)
```
Returns the current source rectangle. The default source rectangle is the image’s natural bounds.

Drawing

```
      ComponentResult GraphicsImportSetGWorld(GraphicsImportComponent
            gi, CGrafPtr port, GDHandle gd)
```
Defines the graphics world for drawing to take place in. Resetting the graphics world is a relatively expensive operation and so should be avoided when possible. This is true even when setting the graphics world to the same graphics world.

```
      ComponentResult GraphicsImportGetGWorld(GraphicsImportComponent
            gi, CGrafPtr *port, GDHandle *gd)
```
Returns the current graphics world for drawing. If the port or gd parameters are not required, they may be set to nil.

```
      ComponentResult GraphicsImportDraw(GraphicsImportComponent gi)
```
GraphicsImportDraw draws the image using the current settings. If GraphicsImportSetGWorld has not been called, the current port will be used.

Saving

Graphics import components allow their data to be saved into two other formats: QuickDraw pictures and QuickTime Image Files. This capability is only needed by programs which perform translation of file formats. Applications that only wish to draw the image can use GraphicsImportDraw.

```
ComponentResult
      GraphicsImportSaveAsPicture(GraphicsImportComponent gi,
      const FSSpec *fss, ScriptCode scriptTag)
```

Creates a new QuickDraw picture file containing the image currently in use by the graphics import component. If possible, the picture will remain in the compressed format. For example, if the image file is a JFIF file, the picture will contain compressed JPEG data. If you don't know what the scriptTag is pass reply.sfScript or smSystemScript.

```
ComponentResult
      GraphicsImportSaveAsQuickTimeImageFile(GraphicsImportCompone
      nt gi, const FSSpec *fss, ScriptCode scriptTag)
```

Creates a new QuickTime Image File containing the image currently in use by the graphics import component. If possible, the data will remain in the compressed format. For example, if the image file is a JFIF file, the QuickTime Image File will contain compressed JPEG data. If you don't know what the scriptTag is pass reply.sfScript or smSystemScript.

Miscellaneous

```
ComponentResult
      GraphicsImportGetDataOffsetAndSize(GraphicsImportComponent
      gi, unsigned long *offset, unsigned long *size)
```

This function returns the offset and size of the actual image data within the file. By default the offset returned is zero and the size returned is the size of the file. However, some graphics import components will override this function to skip over header information.

QuickTime Image Files

Do we really need another image file format? Yes. QuickTime's ability to wrap any compressed data into a QuickDraw picture is a wonderful thing from a compatibility perspective. However, for applications that want to work with the compressed data contained within the picture, it is a real pain. Determining if compressed data is present, and extracting it requires writing obscure code to sit in QuickDraw bottlenecks to spool data and try to catch compressed data running by. Furthermore, there are all sorts of weird cases to deal with like multiple compressed images in a single file. The QuickTime Image File solves this and a few other issues.

The QuickTime Image File is intended to be the most natural way to carry around QuickTime compressed data. The file uses the same atom based structure as a QuickTime movie. There are only two defined atom types, 'idsc' which contains an image description and 'idat' which contains the actual image data. For a JPEG image, the image description atom would contain a QuickTime image description describing the JPEG image's size, resolution, depth, etc. and the image data atom would contain the actual JPEG compressed data. The QuickTime image file can also contain other atoms. For example, it can contain single fork preview atoms as defined in the QuickTime 2.1 documentation. Furthermore, since the QuickTime Image File is a single fork format, it will work well in cross platform areas.

On the Macintosh, the QuickTime Image File has a file type of ‘qtif’.