PICTools™ and AIMTools™

Programmer's Guide

Imaging Development Kits

May 2022

Table of Contents

Introduction

About This Guide

The purpose of this Programmer's Guide is to show how to use PICTools and AIMTools to solve some of the image processing problems facing application developers. These problems include compressing and decompressing images, reading and writing image files, converting between image file formats, transforming images and displaying images. The guide describes the application interface (API) provided by the PICTools and AIMTools toolkits. The PICTools Programmer's Reference and the AIMTools Programmer's Reference documents are companion documents to this Programmer's Guide; they provide extensive specific documentation on the usage of the API.

AIMTools is designed to solve image processing challenges for Developers writing mobile applications. AIMTools is based on the same architecture as the PICTools product, so this Programmer's Guide is relevant to either product. Throughout this Programmer's Guide document, references to PICTools also apply to AIMTools. Both Programmer's Reference documents include a comprehensive list of which opcodes are available in each SDK.

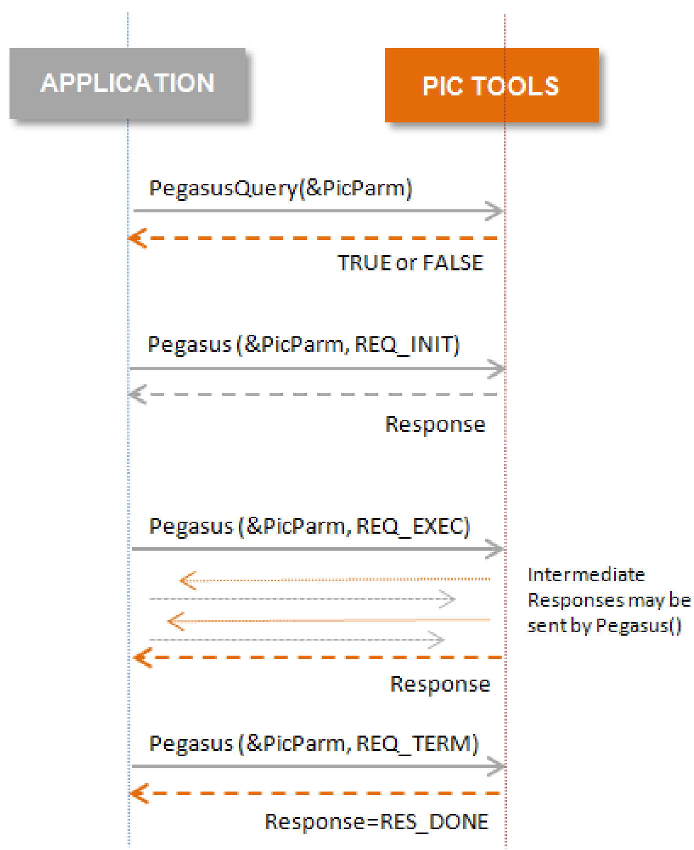
Overview of the PICTools API

Getting Image Information: PegasusQuery()

The PICTools API provides the application with a way to obtain information about an image by calling the **PegasusQuery() function. PegasusQuery()** is passed an input buffer containing image data. It returns information about the image including, for example, the image file format, height and width.

Requesting an Operation to be Performed to an Image: Pegasus()

The main purpose of using PICTools is to perform *operations* on a given image. For example, one operation provided by PICTools is "compress a DIB image to a sequential JPEG image". Another operation is "rotate a sequential JPEG image". An operation is supplied with image data as input. The operation acts on the input image data and produces output image data. The output image is returned to the application requesting the PICTools operation. Operations are also referred to as *opcodes*, because each operation is requested using a manifest constant called an opcode.



An operation is requested by calling a function named **Pegasus**. A REQUEST code and a pointer to a PIC_PARM data structure are parameters to the function and the returned value is a RESPONSE code. The PIC_PARM data eventually includes all the input, output and input/output parameters needed to perform the requested operation.

In the simplest case, **Pegasus** is called three times. The first time it is called, REQ_INIT is the request code and all operation initialization is performed. The second time it is called, REQ_EXEC is the request code and the requested operation acts on the input data, producing the output data. The last time it is called, REQ_TERM is the request code and all operation cleanup occurs. In this simplest case, each call returns a response of RES_DONE if no error was detected or RES_ERR if an error was detected.

Handling Pegasus() Responses

In some scenarios, Pegasus may send one or more intermediate responses to the application before the requested operation is completed and the final response (usually RES_DONE or RES_ERROR) is sent. Examples of common scenarios where intermediate responses are expected include:

a. The application is using an input buffer smaller than the input image. In this case, every time Pegasus consumes all the data in the input buffer, it notifies the application that it needs more data.

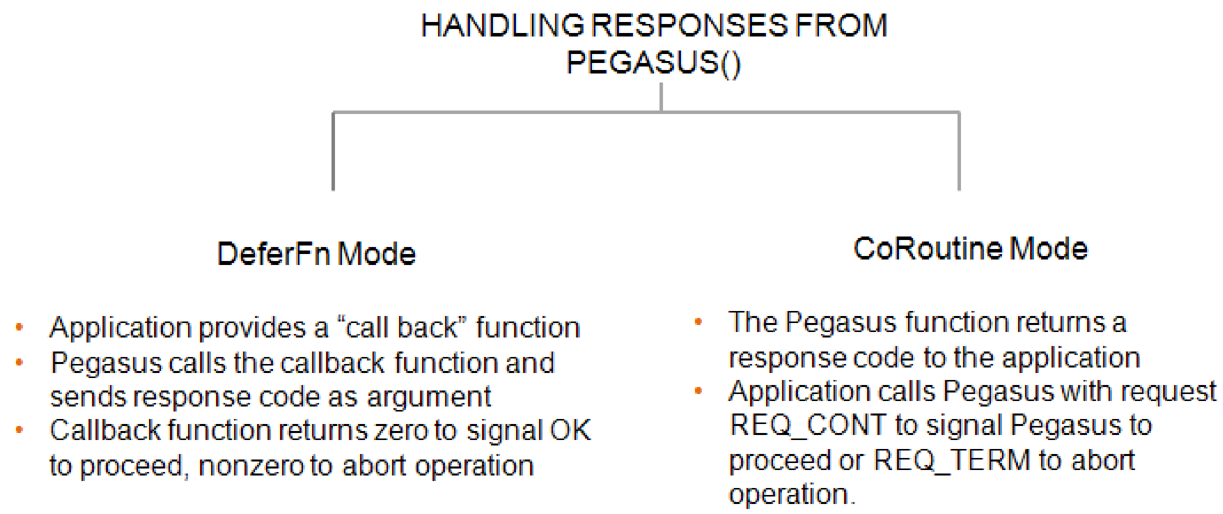
b. The application is using an output buffer smaller than the output image. Once Pegasus fills up the output buffer, it notifies the application that it needs more space for output.

c. The application has asked to be notified whenever some portion of an operation has been performed. As the operation completion proceeds, Pegasus notifies the application about its progress.

In some cases, an intermediate response from Pegasus may require some processing from the application before Pegasus can resume performing the requested operation. In other cases, no response or action by the application is expected and the intermediate response is just a notification.

The application can choose between two different modes of receiving and handling these intermediate responses from Pegasus:

- One mode is the "DeferFn" or callback mode. In this mode, the application provides a response-handling (call back) function pointer in the PIC_PARM data. The function is called by Pegasus to send an intermediate response. The application logic to process the intermediate response is coded in the call back function. A nonzero return value from the call back function to Pegasus is a signal to Pegasus to abort processing. A zero return value indicates that Pegasus should continue processing.
- The other mode is coroutine mode. In this mode, the application's call to Pegasus returns whenever intermediate response is to be sent to the application. The application performs any required action, if needed, and continues processing by calling Pegasus again using a REQ_CONT request parameter or aborts processing by calling Pegasus again using a REQ_TERM request parameter.



Input and Output Parameters: PIC_PARM Structure.

The PIC_PARM structure passed as an argument to the **Pegasus** function is the main mechanism to allow code that is calling PICTools to exchange information with PICTools. PIC_PARM has one set of parameters which is relatively independent of the image operation and another set of parameters which is specific to the image operation.

The set of PIC_PARM parameters independent of the operation can be further subdivided into three categories. The first category is general data. This category includes the PICTools version expected by the application and an opcode specifying the requested operation. The second category is BITMAPINFO data describing the image. The BITMAPINFO data structure is the same data structure which is used in Windows development. The third category specifies the buffering of the input and output data. In the simplest case an input buffer contains all the input data and an output buffer has room for all the output data before **Pegasus** is called.

Note: In many opcodes, the second category is now augmented with a REGION structure. This replaces much of the BITMAPINFOHEADER structure to allow for more general image formats such as .RAW.

The PIC_PARM parameters specific to each operation are encapsulated in a C union with a structure for each type of operation.

```
struct PIC_PARM {
```

General Data

```
 LONG   ParmSize;   /* Size of this structure (bytes) */
 BYTE   ParmVer;    /* API Version for PICTools */
...
```

BITMAP INFO, REGION

```
BITMAPINFOHEADER Head;    /* Uncompressed Image Width, Height, etc. */
RGBQUAD   ColorTable[272]; /* Holds primary (256) and secondary (16) palettes */
...
REGION      RegionIn, RegionOut;
```

Input and Output Buffers

```
QUEUE     Get ;  /* Input Buffer */
QUEUE     Put;   /* Output Buffer */
```

Opcode Specific Data

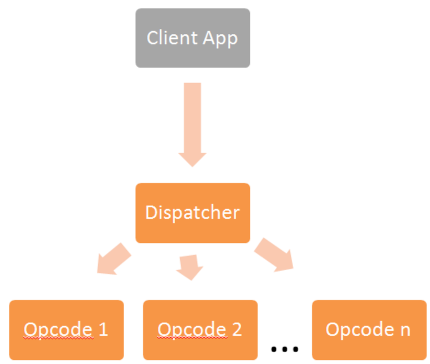
```
union {
      PEGQUERY   QRY;     // PegasusQuery (NOT opcode)

      DIB_INPUT  D2J;      // OP_D2J (progressive or sequential)
      DIB_OUTPUT J2D;     // OP_JE2D, OP_S2D, OP_SE2D
      TRANS2P    S2P;      // OP_S2P
      // etc ...
};
```

```
};
```

1. Brief Overview of the PICTools Architecture

Code using PICTools sits on top of a module called the dispatcher. The dispatcher implements the Pegasus() function as an entry point into the PICTools opcodes. When a particular operation is invoked, the dispatcher finds and loads the appropriate opcode module and calls it. When the opcode returns to the dispatcher, the dispatcher returns to the calling code. The dispatcher contains PegasusQuery, provides common functions needed by the opcodes and facilitates the portability of PICTools across different platforms.



- Dispatcher
  - exposes a common API to clients that is used across all opcodes
  - if using dynamic libraries, loads opcode libraries for execution
  - provides common functions to the opcodes
  - contains PegasusQuery
- Opcodes
  - PICTools functions are grouped into cohesive, modular libraries (opcodes)
  - each opcode implements a specific technology, e.g.
    - sequential JPEG compression opcode
    - binarize opcode
    - JPEG 2000 expand opcode
    - document cleanup opcode
  - modular opcode architecture means that client applications only need to deploy the specific opcode or opcodes containing the functionality required by the application
  - Complete lists of opcodes are provided in the PICTools Programmer's Reference and the AIMTools Programmers' Reference documents.

The following sections describe the PICTools API in more detail.

- **A Code Example** contains example code to expand a sequential JPEG image into a Windows DIB (Device-Independent-Bitmap).
- **Calling Pegasus** describes how to call **Pegasus**, what return values are expected and what application actions are expected based upon the return values.
- **Setting the PIC_PARM Structure** describes how to initialize the PIC_PARM structure before, and in some cases between, the calls to **Pegasus**.
- **Queue Management** describes how to use the buffering options provided by **Pegasus**.

2. A Complete Code Example

Following is a complete example which decompresses a sequential JPEG image to a DIB. The function `ExpandJPEGTo24BitDIB` () receives a buffer which already contains a JPEG image. The function allocates and returns a buffer containing a DIB expanded from the JPEG image. The BITMAPINFO for the DIB is also returned. The function returns ERR_NONE if no error occurs. Otherwise it returns a PICTools API error code[1].

Include Files

All programs using PICTools must include the file pic.h in their code. This file contains the API declarations required by the application and will pull in other PICTools include files as needed. The application may need to include errors.h also, if the application refers to the manifest error constants that represent PICTools error codes. In this example, errors.h is included so that the code can use `ERR_BAD_IMAGE_TYPE, ERR_UNEXPECTED_RESPONSE, etc.` PICTools error codes are returned in PicParm.Status if the RESPONSE code is RES_ERR.

Validating the image type

The example uses **PegasusQuery** to validate that the image type is a sequential JPEG image. The PIC_PARM data structure is always initialized to 0 first. Then specific PIC_PARM fields are initialized for the call to **PegasusQuery**. Specifically, the input buffer pointers are set to point to the JPEG image buffer. See the **Queue Management** section for more information about setting the input buffer pointers. Finally, **PegasusQuery** is called and the returned image type is tested. See the **Setting the PIC_PARM Structure** and **Calling PegasusQuery** sections for additional information.

```
LONG ExpandJPEGTo24BitDIB(
    LPBYTE       pbInputBuffer,       // pointer to JPEG image
    DWORD        dwInputLength,       // length of JPEG image
    LPBYTE      *ppbOutputBuffer,     // receive pointer to DIB
    DWORD       *pdwOutputLength,     // receive length of DIB
    LPBITMAPINFO pOutputBitmapInfo)   // receives DIB BITMAPINFO
{
    PIC_PARM ppquery;  // to be used with PegasusQuery()
    PIC_PARM ppwork;   // to be used with Pegasus()
    RESPONSE response;

    // See PICTools and AIMTools Programmers Guide Section 6
    // for more information on Setting the PIC_PARM Structure.
    // general PIC_PARM initialization for all operations
    memset(&ppquery, 0, sizeof(ppquery));
    ppquery.ParmSize     = sizeof(ppquery);
    ppquery.ParmVer      = CURRENT_PARMVER; // #define'd in PIC.H
    ppquery.ParmVerMinor = 1; // a magic number for the current version

    // initialize input buffer pointers
    ppquery.Get.Start  = pbInputBuffer;
    ppquery.Get.End    = pbInputBuffer + dwInputLength;
```

[1] It also returns ERR_NONE if an invalid registration code error occurs.

```
// initialize input queue pointers
ppquery.Get.Front  = ppquery.Get.Start;
ppquery.Get.Rear   = ppquery.Get.End;
ppquery.Get.QFlags = Q_EOF; // since no input follows


// identify and validate the source image type
// request that PegasusQuery return the image type
ppquery.u.QRY.BitFlagsReq = QBIT_BICOMPRESSION;
if ( !PegasusQuery(&ppquery) ||
    ( ppquery.Head.biCompression != BI_picJPEG &&
      ppquery.Head.biCompression != BI_PICJ ) )
    return ( ERR_BAD_IMAGE_TYPE );

// for this sample we are only going to support 24 bpp RGB
// images, however 8bpp gray and other formats are also supported
// by the opcode.
if ( ppquery.Head.biBitCount != 24 )
    return (ERR_BAD_BIT_COUNT);
```

Pegasus DeferFn mode

In this simple example, **Pegasus** will not need any additional information so the DeferFn function can be minimal:

```
LONG DeferFn(PIC_PARM* pPicParm, RESPONSE response)
{
    // For this trivial example, just return 0 to continue
    // the operation. See sample program source in the samples
    // directories for more detailed and realistic examples of
    // DeferFn() usage.
    NOREF(pPicParm);
    NOREF(response);
    return 0;
}

// initialize DeferFn to handle requests from Pegasus
PicParm.DeferFn = DeferFn;
PicParm.Flags |= F_UseDeferFn;
```

Pegasus initialization phase

Before calling Pegasus to decompress the image, it is necessary to reset the PIC_PARM structure and initialize the operation (calling REQ_INIT). In this case, with PIC_PARM initialized like this, if the initialization call to **Pegasus** returns anything other than a RES_DONE response, an error or unexpected condition has occurred. For additional information see the **Calling Pegasus** section.

```
// set PicParm and initialize Pegasus
memset(&ppwork, 0, sizeof(ppwork));
ppwork.ParmSize     = sizeof(ppwork);
ppwork.ParmVer      = CURRENT_PARMVER; // #define'd in PIC.H
ppwork.ParmVerMinor = 1; // a magic number for the current version
ppwork.Get.Start  = pbInputBuffer;
ppwork.Get.End    = pbInputBuffer + dwInputLength;
```

```
ppwork.Get.Front  = ppwork.Get.Start;
ppwork.Get.Rear   = ppwork.Get.End;
ppwork.Get.QFlags = Q_EOF; // since no input follows
```

```
    // initialize parameters specific to expand JPEG
    ppwork.Op = OP_S2D; // requests Sequential JPEG to DIB operation
    ppwork.u.J2D.DibSize = 24; // request 24-bit DIB output

    // A heavily-compressed decoded image might not look as good as it could
    // but this will be faster.
    ppwork.u.J2D.PicFlags |= PF_NoCrossBlockSmoothing;

   // initialize DeferFn to handle intermediate responses from Pegasus
   ppwork.DeferFn = DeferFn;
   ppwork.Flags |= F_UseDeferFn;

   // JPEG expand operation initialization
   response = Pegasus(&ppwork, REQ_INIT);
   if ( response == RES_ERR )
   {
       // The opcodes OP_S2D and OP_SE2DPLUS will both expand a JPEG file
       // to a bitmap, however OP_S2D may not be available on all
       // platforms. If we get an opcode not found error,  attempt to load
       // the opcode OP_SE2DPLUS and try again.
       if (ppwork.Status == ERR_OPCODE_DLL_NOT_FOUND)
       {
           ppwork.Op = OP_SE2DPLUS;
           ppwork.Status = 0;
           response = Pegasus(&ppwork, REQ_INIT); // try again.
       }
       if ( response == RES_ERR )
           return ( ppwork.Status );
   }

   // any other response is unexpected
   if ( response != RES_DONE )
   {
       Pegasus(&ppwork, REQ_TERM);
       return (ERR_UNEXPECTED_RESPONSE );
   }
```

Allocating the DIB output buffer

During initialization, **Pegasus** computes the length in bytes of each DIB image line according to the image width and the requested output bits per pixel. It is convenient, therefore, to allocate the output DIB buffer after initialization. In many cases the algorithm to compress or decompress an image requires a multiple of lines, not just a single line. For example JPEG compresses or decompresses 8 or 16 lines at a time depending on the input image color subsampling. The minimum such set of lines times the number of bytes per line is called the StripSize. The DIB buffer size must be at least StripSize or it can be larger. If it is larger, it must be an integer multiple of StripSize, or large enough for the full output image. See the **Queue Management** section for additional information about setting the PIC_PARM pointers to the output buffer.

```
   // Allocate DIB output buffer
   *pdwOutputLength = ppwork.Head.biHeight * ppwork.u.S2D.WidthPad;
   *ppbOutputBuffer = malloc(*pdwOutputLength);
   if ( *ppbOutputBuffer == 0 )
   {
       Pegasus(&ppwork, REQ_TERM);
       *pdwOutputLength = 0;
```

```
        return ( ERR_OUT_OF_SPACE );
    }

    // See PICTools and AIMTools Programmers Guide Section 7
    // Queue Management for more information on setting
    // the Put Queue pointers.
    // initialize output buffer pointers
    ppwork.Put.Start   = *ppbOutputBuffer;
    ppwork.Put.End     = *ppbOutputBuffer + *pdwOutputLength;

    // See PICTools and AIMTools Programmers Guide Section 7.3
    // Reversed Linear Buffer.
    // initialize reversed output queue pointers
    ppwork.Put.Front   = ppwork.Put.End;
    ppwork.Put.Rear    = ppwork.Put.Front;
    ppwork.Put.QFlags = Q_REVERSE; // top DIB line is at buffer bottom
```

Pegasus execution phase

Now **Pegasus** is called again to execute the expand operation. In this case, with PIC_PARM initialized like this, if it returns anything other than RES_DONE, some error or unexpected condition has occurred.

```
    // JPEG expand operation execution
    response = Pegasus(&ppwork, REQ_EXEC);

    if ( response == RES_ERR )
    {
        // If Pegasus returns RES_ERR then we do not
        // need to call REQ_TERM
        free(*ppbOutputBuffer);
        *ppbOutputBuffer = 0;
        *pdwOutputLength = 0;
        return ( ppwork.Status );
    }

    // any other response is unexpected
    if ( response != RES_DONE )
    {
        Pegasus(&ppwork, REQ_TERM);
        free(*ppbOutputBuffer);
        *ppbOutputBuffer = 0;
        *pdwOutputLength = 0;
        return ( ERR_UNEXPECTED_RESPONSE );
    }
```

Pegasus cleanup phase

If the expected response RES_DONE was returned, **Pegasus** is called a final time to clean up after the expand operation so that internally allocated memory can be released.. If the response was RES_ERR then cleanup has already been done so **Pegasus** need not be called again. Any memory allocated by the Application can be freed at this time, or subsequently when it is no longer needed by the application.

The PicParm.Head field now contains the BITMAPINFOHEADER describing the output DIB, immediately followed by the DIB's color table (PicParm.ColorTable) where applicable. This information is returned to the caller so that the DIB can be displayed or written to a Windows .BMP file.

```
// JPEG expand operation cleanup
response = Pegasus(&ppwork, REQ_TERM);

// any other response is unexpected
if ( response != RES_DONE )
    return ( ERR_UNEXPECTED_RESPONSE );

// return the DIB bitmap info and color table
memcpy( pOutputBitmapInfo, &ppwork.Head, sizeof(ppwork.Head) );

return ( ERR_NONE );
```

The Complete Code Example

The following example just collects the above code fragments into one place and provides a main() function for the application.

```
//*******************************************************************
// Very simple example showing how to use an opcode.  This code is
// available in the PICTools and AIMTols Programmer's Guide.
// The ExpandJPEGTo24BitDIB function contains all of the relevant
// PICTools code and expands a JPEG file to a Bitmap.
//*******************************************************************

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "pic.h"                    // PICTools API declarations
#include "errors.h"                 // PICTools error codes

#define NOREF(x)  (x=x)             // suppress unreferenced parameter
                                    // compiler warning


// See PICTools and AIMTools Programmers Guide Section 3
// Calling Pegasus for more information about DeferFn.
//********************************************************************
// DeferFn
//********************************************************************
LONG DeferFn(PIC_PARM* pPicParm, RESPONSE response)
{
    // For this trivial example, just return 0 to continue
    // the operation. See sample program source in the samples
    // directories for more detailed and realistic examples of
    // DeferFn() usage.
    NOREF(pPicParm);
    NOREF(response);

    return 0;
}

//********************************************************************
// ExpandJPEGTo24BitDIB
//********************************************************************
LONG ExpandJPEGTo24BitDIB(
    LPBYTE       pbInputBuffer,       // pointer to JPEG image
    DWORD        dwInputLength,       // length of JPEG image
    LPBYTE      *ppbOutputBuffer,     // receive pointer to DIB
    DWORD       *pdwOutputLength,     // receive length of DIB
    LPBITMAPINFO pOutputBitmapInfo)   // receives DIB BITMAPINFO
{
    PIC_PARM ppquery;  // to be used with PegasusQuery()
    PIC_PARM ppwork;   // to be used with Pegasus()
    RESPONSE response;
```

```
// See PICTools and AIMTools Programmers Guide Section 6
// for more information on Setting the PIC_PARM Structure.
// general PIC_PARM initialization for all operations
memset(&ppquery, 0, sizeof(ppquery));
ppquery.ParmSize     = sizeof(ppquery);
ppquery.ParmVer      = CURRENT_PARMVER; // #define'd in PIC.H
ppquery.ParmVerMinor = 1; // a magic number for the current version

// initialize input buffer pointers
ppquery.Get.Start  = pbInputBuffer;
ppquery.Get.End    = pbInputBuffer + dwInputLength;

// initialize input queue pointers
ppquery.Get.Front  = ppquery.Get.Start;
ppquery.Get.Rear   = ppquery.Get.End;
ppquery.Get.QFlags = Q_EOF; // since no input follows

// identify and validate the source image type
// request that PegasusQuery return the image type
ppquery.u.QRY.BitFlagsReq = QBIT_BICOMPRESSION;
if ( !PegasusQuery(&ppquery) ||
    ( ppquery.Head.biCompression != BI_picJPEG &&
      ppquery.Head.biCompression != BI_PICJ ) )
    return ( ERR_BAD_IMAGE_TYPE );

// for this sample we are only going to support 24 bpp RGB
// images, however 8bpp gray and other formats are also supported
// by the opcode.
if ( ppquery.Head.biBitCount != 24 )
    return (ERR_BAD_BIT_COUNT);

// See PICTools and AIMTools Programmers Guide Section 6
// for more information on Setting the PIC_PARM Structure.
// general PIC_PARM initialization for all operations
memset(&ppwork, 0, sizeof(ppwork));
ppwork.ParmSize     = sizeof(ppwork);
ppwork.ParmVer      = CURRENT_PARMVER; // #define'd in PIC.H
ppwork.ParmVerMinor = 1; // a magic number for the current version
ppwork.Get.Start  = pbInputBuffer;
ppwork.Get.End    = pbInputBuffer + dwInputLength;
ppwork.Get.Front  = ppwork.Get.Start;
ppwork.Get.Rear   = ppwork.Get.End;
ppwork.Get.QFlags = Q_EOF; // since no input follows

// initialize parameters specific to expand JPEG
ppwork.Op = OP_S2D; // requests Sequential JPEG to DIB operation
ppwork.u.J2D.DibSize = 24; // request 24-bit DIB output

// A heavily-compressed decoded image might not look as good as it could
// but this will be faster.
ppwork.u.J2D.PicFlags |= PF_NoCrossBlockSmoothing;

// initialize DeferFn to handle intermediate responses from Pegasus
ppwork.DeferFn = DeferFn;
ppwork.Flags |= F_UseDeferFn;
```

```
// JPEG expand operation initialization
response = Pegasus(&ppwork, REQ_INIT);
if ( response == RES_ERR )
{
    // The opcodes OP_S2D and OP_SE2DPLUS will both expand a JPEG file
    // to a bitmap, however OP_S2D may not be available on all
    // platforms. If we get an opcode not found error,  attempt to load
    // the opcode OP_SE2DPLUS and try again.
    if (ppwork.Status == ERR_OPCODE_DLL_NOT_FOUND)
    {
        ppwork.Op = OP_SE2DPLUS;
        ppwork.Status = 0;
        response = Pegasus(&ppwork, REQ_INIT); // try again.
    }
    if ( response == RES_ERR )
        return ( ppwork.Status );
}

// any other response is unexpected
if ( response != RES_DONE )
{
    Pegasus(&ppwork, REQ_TERM);
    return ( ERR_UNEXPECTED_RESPONSE  );
}

// Allocate DIB output buffer
*pdwOutputLength = ppwork.Head.biHeight * ppwork.u.S2D.WidthPad;
*ppbOutputBuffer = malloc(*pdwOutputLength);
if ( *ppbOutputBuffer == 0 )
{
    Pegasus(&ppwork, REQ_TERM);
    *pdwOutputLength = 0;
    return ( ERR_OUT_OF_SPACE );
}

// See PICTools and AIMTools Programmers Guide Section 7
// Queue Management for more information on setting
// the Put Queue pointers.
// initialize output buffer pointers
ppwork.Put.Start   = *ppbOutputBuffer;
ppwork.Put.End     = *ppbOutputBuffer + *pdwOutputLength;

// See PICTools and AIMTools Programmers Guide Section 7.3
// Reversed Linear Buffer.
// initialize reversed output queue pointers
ppwork.Put.Front  = ppwork.Put.End;
ppwork.Put.Rear   = ppwork.Put.Front;
ppwork.Put.QFlags = Q_REVERSE; // top DIB line is at buffer bottom

// JPEG expand operation execution
response = Pegasus(&ppwork, REQ_EXEC);

if ( response == RES_ERR )
{
```

```
        // If Pegasus returns RES_ERR then we do not
        // need to call REQ_TERM
        free(*ppbOutputBuffer);
        *ppbOutputBuffer = 0;
        *pdwOutputLength = 0;
        return ( ppwork.Status );
    }

    // any other response is unexpected
    if ( response != RES_DONE )
    {
        Pegasus(&ppwork, REQ_TERM);
        free(*ppbOutputBuffer);
        *ppbOutputBuffer = 0;
        *pdwOutputLength = 0;
        return ( ERR_UNEXPECTED_RESPONSE );
    }

    // JPEG expand operation cleanup
    response = Pegasus(&ppwork, REQ_TERM);

    // any other response is unexpected
    if ( response != RES_DONE )
        return ( ERR_UNEXPECTED_RESPONSE );

    // return the DIB bitmap info
    memcpy( pOutputBitmapInfo, &ppwork.Head, sizeof(ppwork.Head) );

    return ( ERR_NONE );
}

// Write byte to a file.
PRIVATE void putb(FILE *file, BYTE b)
{
    putc(b,file);
}

// Write WORD in little-endian format to a file.
PRIVATE void putwle(FILE *file, WORD w)
{
    putb(file,(BYTE)w);
    putb(file,(BYTE)(w >> 8));
}

// Write DWORD in little-endian format to a file.
PRIVATE void putdwle(FILE *file, DWORD dw)
{
    putwle(file,(WORD)dw);
    putwle(file,(WORD)(dw >> 16));
}

int main(int argc, char* argv[])
{
    FILE* InputFile = 0;
    FILE* OutputFile = 0;
```

```
LONG ret = 0;
LPBYTE InputBuffer = 0;
DWORD dwInputSize = 0;
LPBYTE OutputBuffer = 0;
DWORD dwOutputSize = 0;
PICSIZET InputRead = 0;
BITMAPINFO BitmapInfo;
int offbits;

if ( argc < 3 )
{
    printf("Usage: quickstartsample [input.jpg] [output.bmp]\n");
    return ( -1 );
}

InputFile = fopen(argv[1], "rb");

if ( InputFile == 0 )
{
    printf("Failed to open %s.\n", argv[1]);
    return ( -1 );
}


OutputFile = fopen(argv[2], "wb");

if ( OutputFile == 0 )
{
    printf("Failed to open %s.\n", argv[2]);
    return ( -1 );
}

fseek(InputFile, 0, SEEK_END);
dwInputSize = ftell(InputFile);
fseek(InputFile, 0, SEEK_SET);

InputBuffer = malloc(dwInputSize);
if ( InputBuffer == 0 )
{
    printf("Failed to allocate input buffer.\n");
    fclose(OutputFile);
    fclose(InputFile);
    return ( -1 );
}

InputRead = fread(InputBuffer, 1, dwInputSize, InputFile);
fclose(InputFile);


// Make sure the bytes read == to the bytes expected.
if (InputRead != dwInputSize)
{
    printf("Failed to read input file %s. Expected %d bytes, read %d\n",
        argv[2],dwInputSize, InputRead);
    fclose(OutputFile);
```

```
    free(InputBuffer);
    return ( -1 );
}

// The interesting PICTools code is in here.
ret = ExpandJPEGTo24BitDIB(InputBuffer, dwInputSize, &OutputBuffer,
    &dwOutputSize, &BitmapInfo);
if ( ret != ERR_NONE )
{
    if (ret == ERR_BAD_BIT_COUNT)
    {
        printf("This example only supports 24bpp images. Please "
               "choose another input file and try again.\n");
    }
    else
        printf("ExpandJPEGTo24BitDIB returned %d\n", (int)ret);

    fclose(OutputFile);
    free(InputBuffer);
    if (OutputBuffer != NULL)
        free(OutputBuffer);

    return ( -1 );
}

// The BITMAPFILEHEADER and BITMAPINFOHEADER must be little-endian in
// the file. This method ensures the file is written correctly
// regardless of the endian-ness of the machine used to produce this
// file.

// Write bitmap file header
offbits = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) +
          BitmapInfo.bmiHeader.biClrUsed * sizeof(RGBQUAD);
putb(OutputFile,'B');                                          // bfType
putb(OutputFile,'M');
putdwle(OutputFile, offbits + dwOutputSize);                   // bfSize
putwle(OutputFile,0);                                          // Reserved1
putwle(OutputFile,0);                                          // Reserved2
putdwle(OutputFile, offbits);                                  // bfOffBits

// Write bitmap info header
putdwle(OutputFile, sizeof(BITMAPINFOHEADER));
putdwle(OutputFile, BitmapInfo.bmiHeader.biWidth);
putdwle(OutputFile, BitmapInfo.bmiHeader.biHeight);
putwle(OutputFile, 1);
putwle(OutputFile, BitmapInfo.bmiHeader.biBitCount);
putdwle(OutputFile, BitmapInfo.bmiHeader.biCompression);
putdwle(OutputFile, BitmapInfo.bmiHeader.biSizeImage);
putdwle(OutputFile, BitmapInfo.bmiHeader.biXPelsPerMeter);
putdwle(OutputFile, BitmapInfo.bmiHeader.biYPelsPerMeter);
putdwle(OutputFile, BitmapInfo.bmiHeader.biClrUsed);
putdwle(OutputFile, BitmapInfo.bmiHeader.biClrImportant);

// Write out the color table
```

```
    if (BitmapInfo.bmiHeader.biClrUsed != 0)
    {
       fwrite(&BitmapInfo.bmiColors, 1,
               BitmapInfo.bmiHeader.biClrUsed * sizeof(RGBQUAD),OutputFile);
    }

    // Write out the remainder of the file
    fwrite(OutputBuffer, 1, dwOutputSize, OutputFile);

    fclose(OutputFile);


    if ( OutputBuffer != NULL )
        free(OutputBuffer);

    free(InputBuffer);

  return 0;
}
```

3. Calling Pegasus

Before calling **Pegasus**, the application initializes the PIC_PARM structure according to the desired image operation (see the **Setting the PIC_PARM Structure** section). The application then calls **Pegasus** with a REQ_INIT request code to perform initialization for the operation. Next the application calls **Pegasus** with a REQ_EXEC request code to perform the operation. Finally, the application calls **Pegasus** with a REQ_TERM request code to clean up as needed.

When the application calls **Pegasus** with request REQ_INIT, **Pegasus** begins initialization for the requested operation, including allocating memory for internal use. When the application calls **Pegasus** with request REQ_EXEC, **Pegasus** begins performing the requested operation. As **Pegasus** initializes or performs the requested operation it may call an application-supplied **DeferFn** function (or may return in coroutine mode) so that the application can perform some action on behalf of **Pegasus** or to notify the application of the occurrence of some event. **Pegasus** calls the **DeferFn** function (or **Pegasus** returns in coroutine mode):

- when more input is needed
- when more space for output is needed
- when data is needed from or put to a non-contiguous location in the input or output stream
- when a color table has been created
- to allow the application to perform processing during the Pegasus operation (e.g. progress reporting)
- to allow the application to allocate space for an image comment, for image application data, or for other data
- to notify the application that a second or later image comment was encountered
- to allow the application to extend the Pic2List
- other events listed in the RESPONSE typedef

The specific event is identified by the response code returned by **Pegasus**. The appropriate application action is described in the **Handling Pegasus Response Codes** sub-section. After taking the appropriate action, the application will ordinarily continue the operation processing by returning 0 from DeferFn. Otherwise the application may abort the operation by setting the PIC_PARM Status field to an error code and returning 1 from DeferFn. In coroutine mode, the application will ordinarily continue the operation by calling **Pegasus** with a REQ_CONT request parameter. Otherwise the application may abort the operation by calling **Pegasus** with a REQ_TERM request parameter.

The following is pseudo-code for a simple example using DeferFn mode:

```
Response = Pegasus(&PicParm, REQ_EXEC);
if (Response == RES_DONE )
{
  if ( PicParm.Put.Front != PicParm.Put.Rear )
    PutSpace(&PicParm);
  Pegasus(&PicParm, REQ_TERM);
}
// else Response == RES_ERR

///////////////////////////////////////////////////
// the DeferFn function might be:
LONG DeferFn(PIC_PARM* pPicParm, RESPONSE Response)
{
  switch ( Response )
  {
    case RES_PUT_NEED_SPACE:
      pPicParm->Status = PutSpace(pPicParm);
```

```
        break;
      case RES_GET_NEED_DATA:
        pPicParm->Status = GetData(pPicParm);
        break;
      default:
        pPicParm->Status = ERR_UNKNOWN_RESPONSE;
       break;
     }
    // return 0 for ERR_NONE, else != 0 and abort
    return ( pPicParm->Status!= ERR_NONE );
  }
```

Pegasus will not return until an error occurs or the operation finishes. As shown above, an application will frequently handle RES_PUT_NEED_SPACE and RES_GET_NEED_DATA responses. However, if the Get buffer contains all input data and Q_EOF is set in **QFlags**, then RES_GET_NEED_DATA will not occur. If the **Put** buffer has space for all output data, then RES_PUT_NEED_SPACE will not occur. See the **Linear Buffer** sub-section of the **Queue Management** section. Note that the code checks the **Put** buffer after the operation is complete to see if there has been additional output. Additional output may have been placed in the **Put** buffer after the last RES_PUT_NEED_SPACE, and that data must also be processed.

An application may receive other responses for certain operations, for certain image types, or if the application has set certain flags or parameters in the PIC_PARM union structure for a requested operation.

Pegasus is multi-instance and thread-safe. This means that a multithreaded application can call **Pegasus** from each thread. Note that in such case, each thread must have its own copy of the PicParm structure. See the Programmer's Reference for additional details regarding calling Pegasus in a multithreaded application.

3.1 PIC_PARM Structure

See the **Setting the PIC_PARM Structure** section.

The PIC_PARM structure cannot be moved.

After **Pegasus** is called with a REQ_INIT request code, subsequent calls to **Pegasus**, until and including the REQ_TERM request, must pass the same PIC_PARM data at the same address.

Application developers must exercise care when performing memory operations with the PicParm data structure (such as copying it or moving it to a different location), because subsequent calls to **Pegasus** as part of the same operation must use the same PIC_PARM data at the same address.

3.2 Table of REQUEST Codes

Code	Pegasus Action
REQ_INIT	Begins initialization appropriate to the requested operation
REQ_EXEC	Begins performing requested operation
REQ_CONT	Continues performing REQ_INIT or REQ_EXEC activities in coroutine mode
REQ_TERM	Terminates and performs cleanup appropriate to the requested operation.

3.3 Table of RESPONSE Codes

Here is a list of some of the most common return codes for Pegasus Responses. A complete list is provided in the PicTools Programmer's Reference and in the AIMTools Programmer's Reference. In each of the following Application Actions, except for RES_DONE and RES_ERR, the application will normally continue the requested operation by returning 0 from its **DeferFn** function or in coroutine mode by calling **Pegasus** with a REQ_CONT request parameter.

Code	Application Action
RES_COLORS_MADE	The application requested that an optimal color table be created. The color table is now available. No action is required.
RES_DONE	The REQ_INIT or REQ_EXEC or REQ_TERM phase of the operation is complete without error. When returned after REQ_INIT, the application calls Pegasus with a REQ_EXEC request. When returned after REQ_EXEC the application will ordinarily call Pegasus with a REQ_TERM request.
RES_ERR	An error was detected and Pegasus has terminated the operation. The error is returned in PIcParm.Status. No action is required and Pegasus does not need to be called with a REQ_TERM request.
RES_EXTEND_PIC2LIST	See the **Accessing Comments and Other Auxiliary Data** section.
RES_GET_DATA_YIELD	The application set the PF_YieldGet flag and Pegasus has processed some amount of input. No action is required.
RES_GET_NEED_DATA	The application must supply additional input data.
RES_HAVE_COMMENT	See the **Accessing Comments and Other Auxiliary Data** section.
RES_NULL_PICPARM_PTR	This is a programming error in the call to Pegasus. The PicParm pointer is the null pointer.
RES_PUT_DATA_YIELD	The application has set the PF_YieldPut flag and Pegasus has output some amount of data. No action is required. It is recommended that you do not modify the PUT queue pointer location in response to this message.
RES_PUT_NEED_SPACE	The application must provide additional space in the output buffer. Ordinarily the application will remove some or all of the data which was placed in the output buffer and will reset the Front and Rear pointers. Note that additional output data may be placed in the output buffer after the final RES_PUT_NEED_SPACE, so you will have to process that output after the operation is complete.
RES_SEEK	The application must provide input or output data starting at the specified offset into the input or output image.

4. Calling PegasusQuery

The **PegasusQuery** function allows various image properties to be obtained from an image. The same PIC_PARM data structure that is used for **Pegasus** is used for **PegasusQuery**.

PegasusQuery uses the PIC_PARM union QRY structure to control which image properties are returned. The QRY structure has a bit-mapped field, PIC_PARM.u.QRY.BitFlagsReq, which controls which image properties are returned. Some image properties are returned in the same QRY structure. Other image properties are returned in the PIC_PARM.Head BITMAPINFOHEADER data structure.

If **PegasusQuery** returns TRUE, then the image data appeared to be valid and all the requested image properties were returned. If FALSE is returned, and the PIC_PARM.Status field is ERR_NONE, then some requested image property was not available. Otherwise **PegasusQuery** returned FALSE and the PIC_PARM.Status field has the error code. The error code is ordinarily ERR_BAD_IMAGE_TYPE indicating that the image is not recognized as a supported image type. Otherwise ERR_NULL_POINTER or ERR_BAD_DATA is returned if the Get queue is unspecified or empty.

If **PegasusQuery** returned FALSE because some image property was not available, you can determine which image properties *were* returned by looking at the PIC_PARM.u.QRY.BitFlagsAck field. If a requested image property was not available, the corresponding bit in BitFlagsAck will be clear.

Following are the possible BitFlagsReq (BitFlagsAck) parameters:

Variable	Value
QBIT_BISIZE	returns PIC_PARM.Head.biSize, ordinarily sizeof(BITMAPINFOHEADER)
QBIT_BIWIDTH	returns PIC_PARM.Head.biWidth, the width or number of colums of the image
QBIT_BIHEIGHT	returns PIC_PARM.Head.biHeight, the height or number of rows in the image
QBIT_BIPLANES	returns PIC_PARM.Head.biPlanes = 1
QBIT_BIBITCOUNT	returns PIC_PARM.Head.biBitCount
QBIT_BICOMPRESSION	returns PIC_PARM.Head.biCompression, the image type
QBIT_BISIZEIMAGE	returns PIC_PARM.Head.biSizeImage, the size in bytes of the image data when decoded.
QBIT_BIXPELSPERMETER	returns PIC_PARM.Head.biXPelsMerMeter
QBIT_BIYPELSPERMETER	returns PIC_PARM.Head.biYPelsPerMeter
QBIT_BICLRUSED	returns PIC_PARM.Head.biClrUsed, the number of colors in the color table
QBIT_BICLRIMPORTANT	returns PIC_PARM.Head.biClrImportant, the number of significant colors in the color table
QBIT_IMAGESIZE	returns PIC_PARM.u.QRY.ImageSize. It contains the ImageSize as the size of the input "file"including header information and image data
QBIT_AUXSIZE	returns PIC_PARM.u.QRY.AuxSize, the size of the buffer needed for auxiliary data in preparation for an image file utility operation
QBIT_NUMIMAGES	returns PIC_PARM.u.QRY.NumImages, the number of images or pages in the file.

QBIT_COMMENT	returns PIC_PARM.Comment if not NULL and PIC_PARM.CommentSize is not 0.
QBIT_PALETTE	returns the image color table, if any, in PIC_PARM.ColorTable
QBIT_SOIMARKER	returns PIC_PARM.u.QRY.SOIMarker, if applicable

Some image file formats support multiple images in the same file. PIC_PARM.u.QRY.ImageNum can be set to indicate the particular image for which properties are to be returned.

Unlike **Pegasus**, **PegasusQuery** has no method for requesting additional input data from the application. Although it doesn't need all of the image data in the input buffer to return image properties, some image properties may not be available when the input buffer contains too little image data.

5. PicTools Libraries.

- On most systems, the dispatcher is available in the form of either a static or dynamic library.
  - The dynamic libraries are found in the SDK in the bin directory
  - The static libraries are found in the SDK in the lib directory
  - The file names for the libraries are as follows:

	Dynamic Library	Static Library
Win32	picn20.dll	picn20m.dll
Win64	picx20.dll	picx20m.dll
Linux32	libpicl20.so	libpicl20.a
Linux64	libpiclx20.so	libpiclx20.a
Solaris-Sparc32	libpicu20.so	libpicu20.a
Solaris-Sparc64	libpicux20.so	libpicux20.a
Solaris-Intel32	libpics20.so	libpics20.a
Solaris-Intel64	libpicsx20.so	libpicsx20.a
Aix32	libpica20.a	n/a
Aix64	libpicax20.a	n/a
OS X 32	libpicmu20.dylib	n/a
OS X 64	libpicmux20.dylib	n/a
iOS	n/a	libpici20.a
Android	libpicd20.so	libpicd20.a

- Opcodes are available as:
  - dynamic libraries
  - SSM files: these are compressed versions of the opcodes that are smaller, load faster, and can be embedded as resources in a client application (see the *PICTools Programmer's Reference* section on PegasusLoadFromRes for details on embedding SSM files)
  - static libraries
  - Some platforms may only allow one of these formats.
  - A separate file exists for each opcode, named with a unique number
  - A complete list of opcodes and opcode numbers can be found in the Overview section of the *PICTools Programmer's Reference*

- The opcode files are provided in the SDK bin directory

- Names for the opcode files are as follows ('??' represents the opcode number):

	Dynamic Library	SSM	Static Library
Win32	picn??20.dll	picn??20.ssm	n/a
Win64	picx??20.dll	picx??20.ssm	n/a
Linux32	n/a	picn??20.ssm	n/a
Linux64	n/a	picx??20.ssm	n/a
Solaris-Sparc32	picu??20.so	n/a	n/a
Solaris-Sparc64	picux??20.so	n/a	n/a
Solaris-Intel32	n/a	picn??20.ssm	n/a
Solaris-Intel64	n/a	picx??20.ssm	n/a
Aix32	pica??20.so	n/a	n/a
Aix64	picax??20.so	n/a	n/a
OS X 32	picm??20.piclib	picn??20.ssm	n/a
OS X 64	n/a	picx??20.ssm	n/a
iOS	n/a	n/a	pici??20.a
Android	picd??20.so	n/a	n/a

5.1 Loading an Opcode DLL

The PICTools dispatcher loads the appropriate opcode when an application calls **Pegasus** for the first time requesting that opcode. By default in Windows, the dispatcher looks for the opcode in the same location as the dispatcher DLL. If the opcode is not found in that location, then the Windows versions of the dispatcher use the Windows directory search order exactly as specified for the Windows **LoadLibrary** function.

5.2 Unloading an Opcode DLL

The normal windows application shutdown procedure unloads all PICTools opcodes. **PegasusUnload** can be used to unload a PICTools DLL prior to application termination if desired.

5.3 Packaging Opcodes into an Application Resource (Windows only)

Applications that desire to have greater control over the version of opcodes that they will load, can package the opcode SSM files provided into a resource file. Then they should use the **PegasusLoadFromRes**() function instead of **PegasusLoad().** Consult the PICTools Programmer's Reference manual for additional details on how to apply this technique.

6. Setting the PIC_PARM Structure

Zero the PIC_PARM Structure

We recommend you "zero" the PIC_PARM structure before setting the fields which are important to your application. Zero is the default for all fields, so if you set everything to zero you can just concentrate on what is important to the requested operation.

Pegasus tests for a non-zero value in the Reserved field. This indicates that the structure has been initialized internally. The Reserved field points to memory which is internally allocated by REQ_INIT, used by REQ_EXEC, and freed by REQ_TERM. If you do not zero the structure, **Pegasus** may believe the operation is already initialized and may try to access the memory pointed to by the Reserved field. This will likely result in an access violation fault, or worse, it might write to an arbitrary area of memory.

6.1 Setting General PIC_PARM Data

The application initializes the following fields before calling **Pegasus** for any operation and before calling **PegasusQuery**.

Field	Value
ParmSize	sizeof(PIC_PARM)
ParmVer	CURRENT_PARMVER
ParmMinorVer	As documented for the opcode
Op	This is the opcode corresponding to requested operation. This field is ignored by PegasusQuery.
DeferFn	In DeferFn mode, this is a pointer to a function to handle requests for additional information from **Pegasus**.
Flags	In DeferFn mode, set the F_UseDeferFn bit so Pegasus will use the DeferFn function. See the PICTools Programmer's Reference for other flag bits and their usage.
Get	See the **Queue Management** section. The Get queue buffer is defined before PegasusQuery. It is ordinarily defined before Pegasus REQ_INIT is called.
Put	See the **Queue Management** section. Some operations require that the Put queue be specified before Pegasus REQ_INIT is called.

6.2 Setting PIC_PARM Operation Data

Each **Pegasus** operation, as well as **PegasusQuery**, uses one of the structures in the PIC_PARM.u union to specify parameters that control the operation. The particular parameters for each operation are described in detail in the Programmer's Reference.

7. Queue Management

7.1 Overview

The Get (input) and Put (output) buffers within the PIC system are implemented in **Pegasus** as circular queues. This powerful data structure provides enough power and flexibility for asynchronous, multi-threaded, insertion and removal of data while the PIC routines operate. However, since not all applications warrant such an elaborate access method, an application will frequently use these queue structures as a simple linear buffer. This document will describe how to use the queues from the simplest linear buffer to the more complicated circular queue.

The queue data structure is defined as:\

```
typedef struct {
  BYTE PICFAR * FrontEnd;
  BYTE PICFAR * Start;
  BYTE PICFAR * Front;
  BYTE PICFAR * Rear;
  BYTE PICFAR * End;
  BYTE PICFAR * RearEnd;
  DWORD         QFlags;
} QUEUE;
```

The FrontEnd and RearEnd pointers only have meaning for a few operations and they are ignored in most applications and in this discussion.

The Start pointer points to the start of a buffer. The End pointer points to one byte past the last byte which is contained in the buffer. Thus (End - Start) is the length of the buffer in bytes. Ignoring the reversed buffer case described later, the Front pointer points to the first byte of valid data in the queue and the Rear pointer points to one byte after the last byte of valid data in the queue. The QFlags are used to indicate queue conditions such as Q_EOF which is used to signal that there will be no more data following the data already present in the queue.

7.2 Linear Buffer

This is the most frequent method for using the queue buffers. A buffer is allocated which is at least big enough to hold all the data. In the following, the buffer is assumed exactly big enough to hold all the data. The Start pointer is set to the start of the buffer and the End pointer is set to end of the buffer -- the address computed by adding the length of the buffer to the address of the start of the buffer.

Empty Linear Buffer

In an empty linear buffer the Front and Rear pointers equal each other. When Front and Rear are equal, they will ordinarily have been reset to the Start of the queue by the code which is supplying the queue with data.

```
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
^                                         ^
Start                                     End
Front
Rear
QFlags = 0
```

where the e's represent empty space.

Full Linear Buffer

In a full linear buffer, the Front pointer equals Start and the Rear pointer equals End.

```
|DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD|
 ^                                ^
 Start                            End
 Front                            Rear
 QFlags = Q_EOF
```

where the D's represent filled space.

Partially Full Linear Buffer

A linear buffer which is neither full nor empty would have one of the following states:

```
|DDDDDDDDDDDDDDDDDDeeeeeeeeeeeeeeeee|
 ^                  ^               ^
 Start              Rear            End
 Front
 QFlags = 0

|eeeeeeeeeeDDDDDDDDDDDDDDDDDDDeeeeeee|
 ^          ^                 ^      ^
 Start      Front             Rear   End
 QFlags = 0

|eeeeeeeeeeeeeeeeeeDDDDDDDDDDDDDDDDDD|
 ^             ^                     ^
 Start         Front                 End
                                     Rear
 QFlags = Q_EOF
```

7.2.1 Linear Get Buffer Processing

In the simplest case all the input is available. Use the buffer containing input data as the Get queue buffer, setting the pointers as in the full buffer case pictured above. For example:

```
PicParm.Get.Start  = pbInputDataBuffer;
PicParm.Get.End    = pbInputDataBuffer + dwInputLengthValidData;
PicParm.Get.Front  = PicParm.Get.Start;
PicParm.Get.Rear   = PicParm.Get.End;
PicParm.Get.QFlags = Q_EOF;
```

In this simplest case, **Pegasus** will not return with a RES_GET_NEED_DATA response.

If all the input isn't available, start with the empty buffer case above as:

```
PicParm.Get.Start = pbInputDataBuffer;
PicParm.Get.End   = pbInputDataBuffer + dwInputBufferAllocatedSize;
PicParm.Get.Front = PicParm.Get.Start;
PicParm.Get.Rear  = PicParm.Get.Front;
```

When **Pegasus** returns with a RES_GET_NEED_DATA response additional input is put into the buffer starting at the Rear pointer. The Rear pointer is advanced past the copied data as:

```
memcpy(PicParm.Get.Rear, pbNewData, dwNewDataLength);
PicParm.Get.Rear += dwNewDataLength;
```

After all the input has been copied to the buffer, and no more input will be available, set the bit in the QFlags field to signal this state as:

```
PicParm.Get.QFlags |= Q_EOF;
```

7.2.2 Linear Put Buffer Processing

In Put buffer processing, the application is consuming data supplied by **Pegasus**. The Put pointers are set to an empty buffer which is large enough to contain all the output produced as:

```
PicParm.Put.Start  = pbOutputBuffer;
PicParm.Put.End    = pbOutputBuffer + dwOutputBufferAllocatedSize;
PicParm.Put.Front  = PicParm.Put.Start;
PicParm.Put.Rear   = PicParm.Put.Front;
```

In this case, **Pegasus** will not return with a RES_PUT_NEED_SPACE response. If **Pegasus** returns with a RES_PUT_DATA_YIELD response, the output is valid from the Front pointer up to, but not including, the Rear pointer. It is recommended that you do not modify the pointer location in response to the RES_PUT_DATA_YIELD message.

```
memcpy(
    pbAnotherBuffer,
    PicParm.Put.Front,
    PicParm.Put.Rear - PicParm.Put.Front);
```

7.3 Reversed Linear Buffer

Device-Independent Bitmaps (DIB's) are stored in reverse order from what you might expect. The top line on the screen is the last line in the DIB buffer. The first line in the DIB buffer, the line at the start of the buffer, is the bottom line on the screen. Most other image formats, for example JPEG, store the image in the order you might expect where the top line on the screen is stored as the first line in the image buffer. A JPEG image is expanded sequentially beginning with the first line in the JPEG image buffer. Therefore, when expanding a JPEG image, the output is produced beginning with the first, and therefore the top, image line.

The problem is that this line is the line which appears at the end of the DIB buffer. In the case we have been discussing, where the output buffer is big enough for the entire DIB, this is only a minor problem because the application must only be aware that the output buffer is valid starting with the end of the buffer and moving towards the start of the buffer, rather than the more usual buffer organization.

A similar problem occurs when compressing a DIB into a JPEG image. The input must be consumed starting from the end of the buffer so that the top image line is compressed first.

When processing these cases, the application sets the Q_REVERSE bit in the QFlags field of the queue data structure to notify **Pegasus** that the buffer organization is reversed in this way.

Unfortunately, things are more complicated than this. First, the DIB doesn't reverse the order of pixels within a line. Thus the last pixel contained in the DIB buffer is the rightmost screen pixel on the top line. A JPEG image also stores the image pixels within a line in that order, left-to-right. Therefore the DIB bytes produced for each individual JPEG image line have to be copied to the buffer in the same order they are produced, rather than in reverse order, but, they have to be copied to the bottom of the empty area in the buffer because the DIB lines are in reverse order.

Finally, it's even a little worse than that because the JPEG image is actually expanded or compressed some number of lines at a time. PICTools refers to this number of lines as a ***strip***. For each strip, if Q_REVERSE is set, **Pegasus** produces output with the strip's bottom screen line at the start of the appropriate area in the buffer and with the strip's top line at the end of the area in the buffer, the same order in which they will appear in the DIB. Therefore, all the bytes in a strip have to be copied to the buffer in the same order they are produced, but at the bottom of the empty area in the buffer. Similar considerations apply if **Pegasus** is consuming DIB data from the Get buffer to produce a JPEG image as output.

Empty Reversed Linear Buffer

In an empty linear buffer the Front and Rear pointers equal each other. When Front and Rear are equal, they will ordinarily have been reset to the End of the queue by the code which is supplying the queue with data.

```
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
e
^                                        ^
Start                                    End
                                         Front
                                         Rear
QFlags = 0
```

Full Reversed Linear Buffer

In a full linear buffer, the Front pointer equals End and the Rear pointer equals Front

```
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
^                                        ^
Start                                    End
Rear                                     Front
QFlags = Q_EOF
```

Partially Full Reversed Linear Buffer

A linear buffer which is neither full nor empty would have one of the following states:

```
eeeeeeeeeeeeeeeeeeDDDDDDDDDDDDDDDDDDD
^                ^                       ^
Start            Rear                    End
                                         Front
QFlags = 0

eeeeeeeeeeeDDDDDDDDDDDDDDDDDDDeeeeeee
^         ^                       ^      ^
Start     Rear                    Front  End
QFlags = 0

DDDDDDDDDDDDDDDDDDDeeeeeeeeeeeeeeeeee
^                      ^                 ^
Start                  Front             End
Rear
QFlags = Q_EOF
```

Note that when the linear buffer is reversed, the Rear pointer points to the last byte of valid data. The Front pointer points one byte past (in the direction towards the End address) the first byte of valid data.

7.3.1 Reversed Linear Get Buffer Processing

In the simplest case, where all the input is available, use the buffer containing input data as the buffer, setting the pointers as in the full buffer case pictured above. For example:

```
PicParm.Get.Start  = pbInputDataBuffer;
PicParm.Get.End    = pbInputDataBuffer + dwInputLengthValidData;
PicParm.Get.Front  = PicParm.Get.End;
PicParm.Get.Rear   = PicParm.Get.Start;
PicParm.Get.QFlags = Q_REVERSE | Q_EOF;
```

In this simplest case, **Pegasus** will not return with a RES_GET_NEED_DATA response.

When all the input isn't available, start with the empty buffer case above as:

```
PicParm.Get.Start  = pbInputDataBuffer;
PicParm.Get.End    = pbInputDataBuffer + dwInputBufferAllocatedSize;
PicParm.Get.Front  = PicParm.Get.End;
PicParm.Get.Rear   = PicParm.Get.Front;
PicParm.Get.QFlags = Q_REVERSE;
```

When **Pegasus** returns with a RES_GET_NEED_DATA response (or RES_GET_DATA_YIELD if appropriate), and the lastmost strip(s) of additional input is (are) available, the strip(s) can be placed into the Get buffer as follows:

```
PicParm.Get.Rear -= dwNewDataAvailableLength;
memcpy(
  PicParm.Get.Rear,
  pbAnotherBuffer,
  dwNewDataAvailableLength);
```

dwNewDataAvailableLength must be an integer multiple of StripSize. Get.End – Get.Start must be an integer multiple of StripSize. After all the input has been copied to the buffer, and no more input will be available, set the QFlags to signal this state as:

```
PicParm.Get.QFlags |= Q_EOF;
```

7.3.2 Reversed Linear Put Buffer Processing

In Put buffer processing, the application is consuming data supplied by **Pegasus**. The Put pointers are set to an empty buffer which is large enough to contain all the output produced as:

```
PicParm.Put.Start  = pbOutputBuffer;
PicParm.Put.End    = pbOutputBuffer + dwOutputBufferAllocatedSize;
PicParm.Put.Front  = PicParm.Put.End;
PicParm.Put.Rear   = PicParm.Put.Front;
```

In this case, **Pegasus** will not return with a RES_PUT_NEED_SPACE response. If a RES_PUT_DATA_YIELD response is returned, the application can, but doesn't have to, make use of the data extending from the Front pointer to the Rear pointer as:

```
memcpy(
    pbAnotherBuffer,
    PicParm.Put.Rear,
    PicParm.Put.Front - PicParm.Put.Rear);
```

The Front pointer would ordinarily *not* be decremented (moving towards the buffer Start) past the copied data. This is because, when Front and Rear are equal, **Pegasus** will reset them both to the End of the buffer. The next output produced would then overwrite the previous data. It is recommended that you do not change the PUT queue pointers in response to this message.

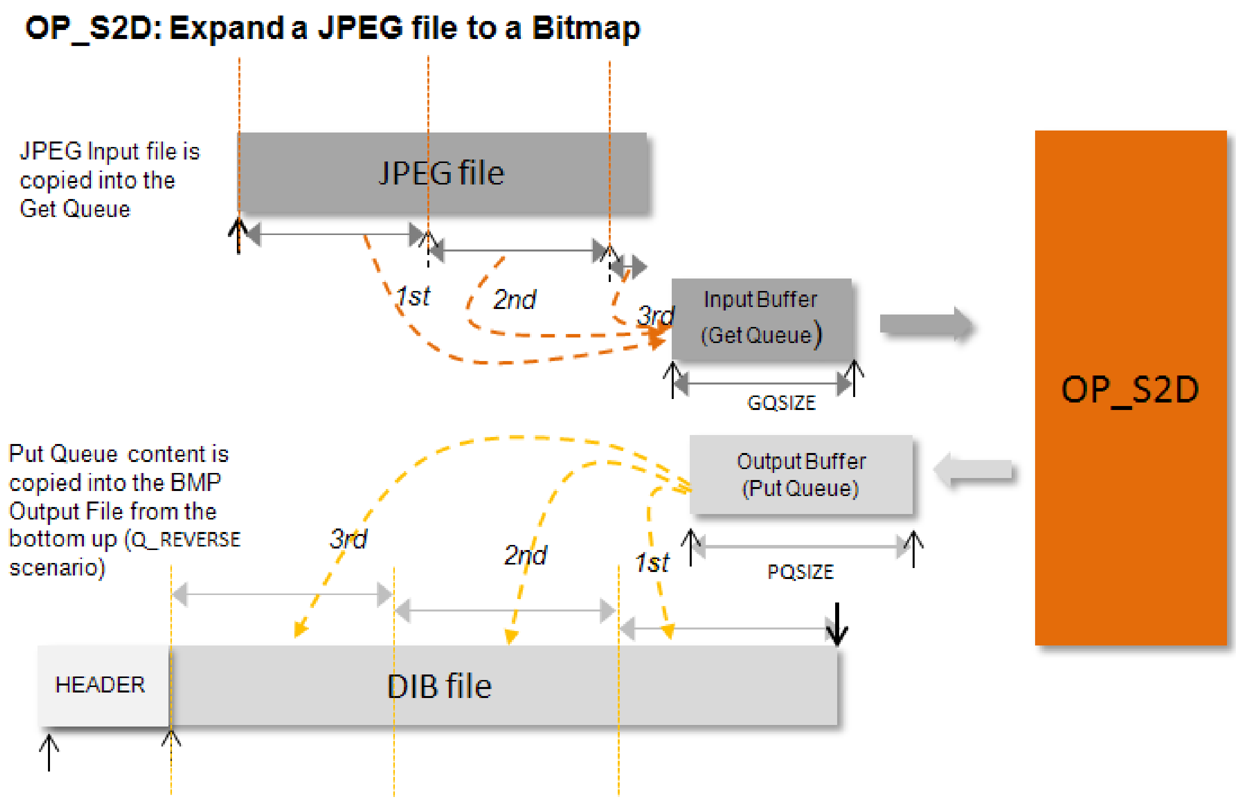
7.4 Sample Code for Linear and Reversed Buffer Processing

This section presents sample application code to illustrate buffer processing in 2 common scenarios: Expanding a JPEG file to a DIB and Compressing a DIB into a JPEG file. It is assumed that the buffer is smaller than the entire image. The code consists of 2 functions. The first function, `CopyFromFileToGetQueue`, is called by the application to pull data from the input file and into the Pegasus input buffer (Get Queue). It would be normally called in response to a `RES_GET_NEED_DATA` reply from Pegasus. The second function, `CopyFromPutQueueToFile`, is called by the application to write data to the output file from the Pegasus output buffer (Put Queue). Normally, this function would be called in response to a `RES_PUT_NEED_SPACE` reply from Pegasus. The sample functions implement the required logic for both the linear and reversed linear queue processing, and determine the appropriate case by checking if the PIC_PARM flag `Q_REVERSE` has been set or not.

Before showing the code, a brief description of the scenarios is provided.

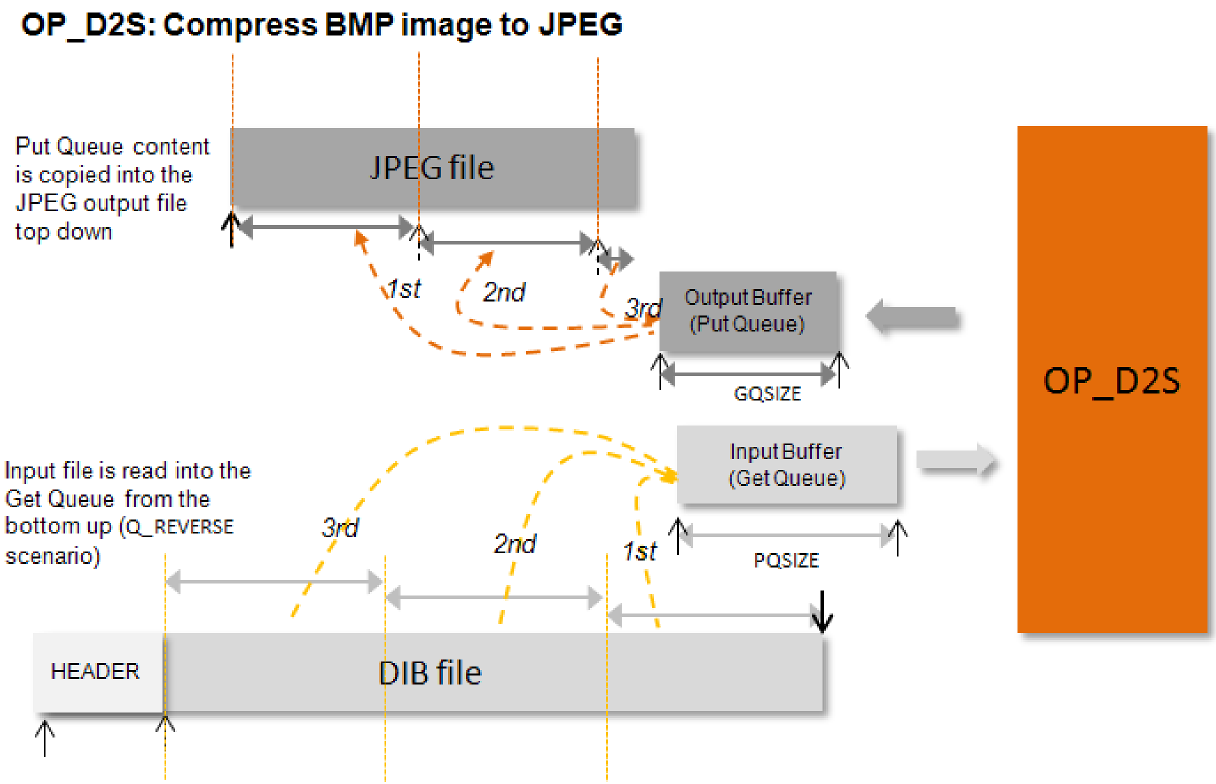
Expanding a JPEG file to a DIB

- This is the typical scenario when using the OP_S2D opcode.
- Requires **linear queue processing** for passing the input image data from the JPEG file into Pegasus.
- Requires **reversed linear queue processing** to save the Pegasus output into the new DIB file.



Compressing a DIB into a JPEG file

- This is the typical scenario when using the OP_D2S opcode.
- Requires **reversed linear queue processing** for passing the input image data from the DIB file into Pegasus.
- Requires **linear queue processing** to save the Pegasus output into the new JPEG file.



Defer Function

- Here is a very simplified implementation of a "Defer Function" that calls `CopyFromFileToGetQueue` and `CopyFromPutQueueToFile`.

```
LONG DeferFn2(PIC_PARM* pp, RESPONSE response)
{
  switch ( response )
   {
     case RES_GET_NEED_DATA:
        CopyFromFileToGetQueue (InputFile, pp);
        break;
     case RES_PUT_NEED_SPACE:
        CopyFromPutQueueToFile (OutputFile, pp);
        break;
   }
   return ( 0 );   // continue the operation
}
```

Sample Code

These functions illustrate just a portion of the application processing needed to support these scenarios. The main items not shown by the sample code are:

- Opening of the input and output files.
- Allocation of the Input and Output buffers. In particular the buffer that will be used to store reversed DIB image data needs to be sized as a multiple of Stripe as mentioned previously.
- In the case of a top down DIB output file, the application needs to write the header information to the file before calling Pegasus (which in turn will eventually call a Defer function that will call `CopyFromPutQueueToFile`). Also the application needs to fseek the file pointer to point to the end of the file before calling Pegasus (see comments for function `CopyFromPutQueueToFile`).

```
/***********************************************************************\
This function would normally be called when replying to a
RES_GET_NEED_DATA response from Pegasus. The input parameters are:
   FILE *fp:    Input file opened for reading.
   PIC_PARM *pp: The PIC_PARM parameter used to communicate with Pegasus

   Just before calling Pegasus (and therefore before this function is called)
   the file pointer should be positioned (fseek) by the application to
   point to the beginning of the image data (0 for JPEG file, file length
   for a top down BMP). From that point on, the application should not
   perform operations that change the file or move the file pointer.
\***********************************************************************/
void CopyFromFileToGetQueue (FILE *fp, PIC_PARM *pp)
{
    LONG nbneeded; // number of bytes needed to fill the input buffer
    LONG nbread;   // number of bytes actually read from the file

    //  Usually a JPEG image is top-down and a BMP image is bottom-up
    if (pp->Get.QFlags & Q_REVERSE) {
        if (pp->Get.Rear == pp->Get.Front) // if input buffer empty:
            pp->Get.Rear = pp->Get.Front = pp->Get.End;

        nbneeded = (LONG) (pp->Get.Rear - pp->Get.Start);

        pp->Get.Rear -= nbneeded;
        fseek(fp, -nbneeded, SEEK_CUR);

        nbread = (LONG)fread(pp->Get.Rear, 1, nbneeded, fp);
        fseek(fp, -nbneeded, SEEK_CUR);
   } else {  // forward (not reversed) scenario
        if (pp->Get.Rear == pp->Get.Front) // if input buffer empty:
            pp->Get.Rear = pp->Get.Front = pp->Get.Start;

        nbneeded = (LONG) (pp->Get.End - pp->Get.Rear);

        nbread = (LONG)fread(pp->Get.Rear, 1, nbneeded, fp);
        pp->Get.Rear += nbread;
    }
    if ( feof(fp) )
        pp->Get.QFlags |= Q_EOF;
    else if ( ferror(fp) )
        pp->Get.QFlags |= Q_IO_ERR;
```

```
}


/**********************************************************************\
This function would normally be called when replying to a
RES_PUT_NEED_SPACE response from Pegasus. The input parameters are:
   FILE *fp:    Input file opened for reading.
   PIC_PARM *pp: The PIC_PARM parameter used to communicate with Pegasus

   Just before calling Pegasus (and therefore before this function is called)
   the file pointer should be positioned (fseek) by the application to
   point to the beginning of the image data. In the forward (not reversed)
   case, such as when saving to a JPEG file, this is just the begining of
   the file.

   In the reversed case, the position should be just past the size
   of the image plus the image header... the preparatory code, would
   look similar to this:
       // populate BitmapHeader and BitmapInfo...
       outputImgPos = sizeof(BitmapHeader) + BitmapInfo.bmiHeader.biSize +
                ppwork.Head.biHeight * ppwork.u.S2D.WidthPad;
       fwrite(&BitmapHeader, 1, sizeof(BitmapHeader), OutputFile);
       fwrite(&BitmapInfo, 1, BitmapInfo.bmiHeader.biSize, OutputFile);
       fseek(OutputFile, outputImgPos, SEEK_SET);

   Note also that once Pegasus is invoked, the application should not
   perform operations that change the file or move the file pointer.
\**********************************************************************/
void CopyFromPutQueueToFile (FILE *fp, PIC_PARM *pp)
{
    LONG nbytesInQ;  // number of bytes on the output buffer, waiting
                     // to be written to file
    LONG nwritten;    // number of bytes actually written to the file

    nbytesInQ = (LONG)(pp->Put.Rear - pp->Put.Front);
    if (nbytesInQ == 0) // is the output buffer empty?
        return;

    //  Usually a JPEG image is top-down and a BMP image is bottom-up
    if (pp->Put.QFlags & Q_REVERSE) {
        nbytesInQ = -nbytesInQ;
        fseek(fp, -nbytesInQ, SEEK_CUR);
        nwritten = (LONG) fwrite(pp->Put.Rear, 1, nbytesInQ, fp);
        if (nwritten != nbytesInQ) {
            pp->Put.QFlags |= Q_IO_ERR; // wrote fewer bytes than attempted
            return;
        }
        fseek(fp, -nbytesInQ, SEEK_CUR);
        pp->Put.Rear = pp->Put.Front = pp->Put.End;
    } else {  // forward (not reversed) scenario
        nwritten = (LONG) fwrite(pp->Put.Front, 1, nbytesInQ, fp);
        if (nwritten != nbytesInQ) {
            pp->Put.QFlags |= Q_IO_ERR; // wrote fewer bytes than attempted
            return;
```

```
        }
        pp->Put.Rear = pp->Put.Front = pp->Put.Start;
    }
}
```

8. Accessing Comments and Other Auxiliary Data

The PICTools libraries provide access to additional data that is frequently stored with the image. Some image formats allow image comments to be stored with the image. Other image formats, such as JPEG, allow application data to be stored with the image. This application data can have any meaning and format which is helpful to the application which created the image but it is not otherwise needed to decode the image. In addition, some PICTools image file utility operations allow auxiliary data to be stored with some image formats. For additional information about auxiliary data, refer to the Programmer's Reference.

The PICTools libraries allow this additional data to be supplied to most operations that output image formats which support the data. The PICTools libraries allow this additional data to be retrieved from most operations that input image formats which support the data.

The comment, application, or other data fields are normally retrieved during the initialization (REQ_INIT) call to **Pegasus** or sometimes via a call to **PegasusQuery**. Comments may also be encountered during REQ_EXEC procession. In the following, retrieving comments and application data is discussed. Similar considerations apply to retrieving auxiliary data.

Newer PICTools operations organize this data as a **PIC2List**. **PIC2List** is a buffer consisting of sequential data packets containing different types of data. For example a comment packet might be followed by an application data packet which might be followed by a second comment packet. The **PIC2List** pointer points to this buffer, which is allocated and freed by the application. **PIC2ListSize** is the size of the buffer in bytes. **PIC2ListLen** is the length of the buffer which contains valid data.

Full support for **PIC2List** has been introduced at different points in time for different opcodes. Please consult the Programmer's Reference for additional details regarding each specific opcode and minimum software version required to use **PIC2List**.

8.1 PIC2List Data Packet

Each PIC2List data packet consists of an 8-bit packet type followed by a 32-bit packet length, followed by the packet data. The packet length specifies the number of bytes of packet data so the total length of the packet including all data is 5 plus the packet length. A 0 byte denoting end-of-packets follows the final packet. The packet types are defined in PIC2FILE.H. Some common types are:

Packet Type	Description
P2P_Comment	ASCIIZ image comment or description.
P2P_RawData	Application-determined binary data.
P2P_Watermark	ASCIIZ image watermark.
P2P_Script	ASCIIZ script for image playback – content is reserved.

8.1.1 PIC2List - Included in Output Image

To include PIC2List comments, etc. in the output image (e.g. comments when compressing a DIB into a JPEG image), set **PIC2List** to point to a buffer for the PIC2List data before calling **Pegasus(REQ_INIT)**. At the same time, set **PIC2ListLen** to the total length of the data -- including a 0 byte marking the end of the packets. **PIC2ListSize** will be ignored by the operation. The following code fragment would put a comment, a watermark, another comment and some application data into the **PIC2List** buffer:

```
#include "pic2file.h"

. . .
{
  P2PktGeneric *p = (P2PktGeneric *)PicParm.PIC2List;
```

```
    P2PktRawData *prd;

    PicParm.ParmVerMinor = 2; /* required for operations supporting
                                 PIC2List */
    p->Type = P2P_Comment;
    p->Length = strlen(FirstComment) + 1; /* include null terminator */
    strcpy(p->Data, FirstComment);
    p = (P2PktGeneric *)( (LPBYTE)p + p->Length + sizeof(P2PktNull) );

    p->Type = P2P_Watermark;
    p->Length = strlen(Watermark);
    strcpy(p->Data, Watermark);
    p = (P2PktGeneric *)( (LPBYTE)p + p->Length + sizeof(P2PktNull) );

    p->Type = P2P_Comment;
    p->Length = strlen(SecondComment) + 1;
    strcpy(p->Data, SecondComment);
    p = (P2PktGeneric *)( (LPBYTE)p + p->Length + sizeof(P2PktNull) );

    prd = (P2PktRawData *)p;
    prd->Type = P2P_RawData;
    prd->Length = sizeof(P2PktRawData) - 5 + AppDataLen;
    memcpy(prd->RawDescription, "APPL", sizeof(prd->RawDescription));
        /* "APP0" .. "APP9", "APPA" .. "APPF" suggested for JPEG,
            all nulls or "APPL" suggested for IMStar */
    prd->RawLength = AppDataLen;
    memcpy(prd->RawData, AppData, AppDataLen); /* raw binary data */
    p = (P2PktGeneric *)( (LPBYTE)p + p->Length + sizeof(P2PktNull) );
    PicParm.PIC2ListLen = (LPBYTE)p - PicParm.PIC2List;
    PicParm.PIC2List[PicParm.PIC2ListLen++] = 0; /* end-of-packets */
}
```

8.1.2 PIC2List - Retrieved from Input Image

8.1.2.1 Application Pre-allocates a Buffer before Pegasus Operation

To discard all comments, etc. from the input image, set **PIC2ListSize** to 0. Otherwise, to retrieve comments from the input image, set **PIC2ListSize** to the allocated size of a buffer, set **PIC2List** to point to the buffer and set **PIC2ListLen** to 0. So long as the buffer is large enough for the comment or other data including an end-of-buffer null byte, an appropriate PIC2List packet will be constructed and copied into the buffer. **PIC2ListLen** will be updated at the time the data is copied, to reflect the data currently in the **PIC2List** buffer.

If the buffer is not large enough for some comment or other data, because the difference between **PIC2ListSize** and **PIC2ListLen** is smaller than needed for the data including the 5-byte packet overhead (and including the end-of-packet byte if **PIC2ListLen** is 0), then a RES_EXTEND_PIC2LIST response is returned from **Pegasus**. This response is handled as described in the following section 9.1.2.2.

8.1.2.2 Application (re)Allocates a Buffer during Pegasus Operation

An application can also avoid allocating a buffer until and unless needed, and can allocate only the specific size needed to retrieve the comment or other data. If the application sets **PIC2ListSize** to -1, then no **PIC2List** buffer need be allocated, but the **Pegasus** will return a RES_EXTEND_PIC2LIST response whenever a comment or other data is encountered.

When RES_EXTEND_PIC2LIST is returned, **PIC2ListLen** is set to the new **PIC2ListSize** required for the buffer in order to retrieve the entire comment and including all packet overhead. **PacketType** is set to the PIC2List packet type so the application can choose to retrieve data only from certain types of packets.

Ordinarily, the application will allocate or reallocate the **PIC2List** buffer and set **PIC2ListSize** to the new size before returning 0 from **DeferFn** as follows:

```
LONG DeferFn(PIC_PARM* pPicParm, RESPONSE response)
{
  switch ( response )
  {
  ..
  case RES_EXTEND_PIC2LIST:
    pNew = realloc(PicParm.PIC2List, PicParm.PIC2ListLen);
    if ( pNew == 0 )
    {
      pPicParm->Status = ERR_OUT_OF_SPACE;
      return ( 1 );  // abort the operation
    }
    PicParm.PIC2List = pNew;
    PicParm.PIC2ListSize = PicParm.PIC2ListLen;
    break;
  ..
  }
  return ( 0 );  // continue the operation
}
```

If there is insufficient space for at least the packet type and packet length on return from **DeferFn**, then no part of the packet is retrieved. So long as **PIC2ListSize** is not 0, any further comments or other data will result in another RES_EXTEND_PIC2LIST response.

8.1.2.3 Accessing Retrieved Data

After RES_DONE is returned from **Pegasus(REQ_EXEC)**, or at any other time that **PIC2ListLen** is not 0, comment and other data may be retrieved by stepping through the PIC2List packets selecting packets of interest and accessing those packets' data. For example, to output image comments to standard output:

```
#include "pic2file.h"

{
P2PktComment *p = PicParm.PIC2List;

if ( PicParm.Pic2ListLen != 0 )
{
  while ( p->Type != P2P_EOF )
  {
    if ( p->Type == P2P_Comment )
      puts(p->Comment);
    p = (P2PktComment*)( (LPBYTE)p + p->Length + sizeof(P2PktNull));
  }
}
}
```

Memory allocated by the application for use in the PIC2List is freed by the application after RES_DONE is returned from **Pegasus(REQ_TERM)**,

9. Using the PIC Libraries

9.1 Include files

Applications using the PICTools API include the PIC.H file, and ordinarily include the ERRORS.H file. PIC2FILE.H and PIC2LIST.H are convenient for **PIC2List** processing. Additional PICTools include files may be needed for some applications and some are included by the above include files.

9.2 Preprocessor Definitions

Applications using the PICTools API must specify the required preprocessor symbols in their projects. The preprocessor symbols required per platform are specified in the following table. The symbols to be used are specified in the column headings.

	WIN32	**WIN64**	**WINDOWS**	**PIC64**	**__unix__**	**__sun__**	**__SPARC__**	**_POWER**	**__AIX64__**
Windows32	X		X						
Windows64	X	X	X	X					
Linux32					X				
Linux64				X	X				
Solaris-Intel32					X	X			
Solaris-Intel64				X	X	X			
Solaris-Sparc32					X		X		
Solaris-Sparc64				X	X		X		
AIX32					X			X	
AIX64				X	X			X	X
Mac-Intel32					X				
Mac-Intel64				X	X				
iOS				++	X				
Android					X				

++ Special note for iOS. An iOS application using the AIMTools API can be built for individual 32-bit or 64-bit architectures, or as a single, universal binary that supports multiple architectures. The AIMTools opcodes for iOS are being updated to be universal binaries for multiple 32-bit and 64-bit architectures. Applications that run on 64-bit architectures (such as arm64 or x86_64) must define the preprocessor symbol PIC64. Those running on 32-bit architectures (such as i386 or armv7) must not define PIC64. A suggested technique for developing common code for either architecture that satisfies this AIMTools requirement automatically is to conditionally define PIC64 according to architecture within the application's precompiled prefix header file. When the compiler defines the symbol __LP64__, ensure that PIC64 is defined as well and not otherwise. Refer to the AIMTools sample programs that demonstrate this practice in their supplied precompiled prefix header files.

9.3 Windows (32-bit, 64-bit) Import Libraries

When using one of the Microsoft compilers to create a 32-bit application using PICN20.DLL, link to the PICNM.LIB import library. When using one of the Microsoft compilers to create a 32-bit application using a static link library, link to the PICN20M.LIB import library. If you are using a Microsoft compiler prior to Visual Studio 2005 use PICN20MVS6.LIB

If using a compiler from another vendor to create a 32-bit application, use PICN.LIB. The PICNM.LIB is a COFF-format library as required by Microsoft's development tools. The PICN.LIB is an OBJ-format library as required by some other vendors' development tools.

When using the Microsoft compilers to create a 64-bit application using PICX20.DLL, link to the PICXM.LIB import library. When using one of the Microsoft compilers to create a 64-bit application using a static link library, link to the PICXM20.LIB import library

9.4 Other platforms: Linux, Solaris, AIX, OSX, iOS, Android

Consult the document "Deploying PicTools applications on Linux, Solaris (SPARC and x86), AIX and OS/X" provided in the documentation directory (doc) of the PicTools SDK. For iOS, consult the "AIMTools iOS QuickStart Guide". For Android, consult the "AIMTools Android QuickStart Guide".

10. Debugging, Tracing and Logging

Additional help is available for PICTools API debugging and problem detection/identification in all environments. The debug dispatcher provides additional parameter validation for **Pegasus**. It also allows tracing and logging of **Pegasus** parameters and responses to a file and/or to the debug monitor.

The debug dispatcher is controlled by a PDEBUG.INI file which must be edited and placed in the Windows directory in Win32 and Win64. The PDEBUG.INI file can have a [General] section with general debug options, an [All] section with options which are defaults for all opcodes, and a section with options for each opcode named with the opcode number (e.g. [10] would contain options for sequential JPEG pack operations). The options in the opcode section override the options in the [All] section for that opcode.

The [General] section can have the following variables:

Variable	Value
LogFile	Names the log file to be used. If the value is empty or the variable is not present, then no log file is used.
AppendLog	If FALSE or unspecified, a new application's log entries overwrite a previous application session's log entries. If TRUE, a new application's log entries are appended to the log file.
FlushLog	If FALSE or unspecified, log entries may not all be written to the disk until the application terminates. If TRUE, then log entries are written to the disk immediately.
DebugMonitor	If FALSE or unspecified, log entries are not displayed on the Debug monitor or device. If TRUE, then log entries are displayed on the Debug monitor or device.
DllName	If TRUE, then log entries contain the opcode DLL name.
TimeStamp	If TRUE, then log entries contain a date/time stamp.
Instance	If TRUE, then log entries assign a unique instance number to each open PIC_PARM for an individual opcode and log entries contain this instance number.
PicParm	If TRUE, then general PIC_PARM fields are reported.
GetQueue	If TRUE, then Get queue pointers are reported.
PutQueue	If TRUE, then Put queue pointers are reported.
PicBitmapInfoHeader	If TRUE, then the Head field in PIC_PARM is reported
PicUnion	If TRUE, then the opcode-specific data in the opcode's union structure is reported for many opcodes.
UseLoadLibrary	This is a Windows and Solaris Sparc setting. If true, the dispatcher loads the opcodes using LoadLibrary (Win32/Win64) or dlOpen (Solaris Sparc32/Sparc64)

The [All] and [<*opcode*>] sections can have the following variables:

Variable	Value
Warning	If TRUE, then warnings are logged.
Error	If TRUE, then errors are logged.
FYI	If TRUE, then other useful information is logged.
Enter	If TRUE, then each entry to Pegasus, PegasusLoad and PegasusUnload is logged.
Exit	If TRUE, then each return from Pegasus, PegasusLoad and PegasusUnload is logged.

10.1 Generating PICTools Debug Log Files on Win32/64

1) In the folder from which the application loads the dispatcher library file, find the dispatcher file: picn20.dll for win32, picx20.dll for win64. Replace the existing file with the debug version (picn20d.dll for win32 and picx20d.dll) renamed to picn20.dll or picx20.dll.
2) Edit the pdebug.ini file from the SDK's bin folder to modify the line "logfile = c:\logfile.txt" to reflect a valid location in your system to write the log file.
3) Place the edited pdebug.ini file in your "WINDIR" folder
4) Run your executable to create the log file named in the pdebug.ini entry for logfile=.
5) Remember to reset the dispatcher library file back to the "release" version when you are finished creating the desired log files.

10.2 Generating PICTools Debug Log Files on Linux, Solaris SPARC/x86, AIX, and OS X

1) If opcodes are not in the /usr/local/lib/pegasus folder, set the environment variable SSMPATH to point to the folder holding the opcodes and export SSMPATH so it is active for the shell and its children.

2) For Linux, replace the existing dispatcher (libpicl20.so or libpiclx20.so in the location added to LD_LIBRARY_PATH above) with libpicl20d.so or libpiclx20d.so renamed (this is important) to libpicl20.so or libpiclx20.so. Alternatively, link your application to the static library dispatcher libpicl20d.a or libpiclx20d.a.

   For Solaris SPARC, this is libpicu20d.so or libpicux20d.so renamed to libpicu20.so or libpicux20.so and copied over the existing dispatcher (in the location added to LD_LIBRARY_PATH above). If linking to the static dispatcher, change " –lpicu20" or " -lpicux20" in the linker flags to " –lpicu20d" or " –lpicux20d" instead.

   For Solaris x86, this is libpics20d.so or libpicsx20d.so renamed to libpics20.so or libpicsx20.so and copied over the existing dispatcher (in the location added to LD_LIBRARY_PATH above). If linking to the static dispatcher, change " –lpics20" or " -lpicsx20" in the linker flags to " –lpics20d" or " –lpicsx20d" instead.

   For AIX, this is libpica20d.a or libpicax20d.a renamed to libpica20.a or libpicax20.a and copied over the existing dispatcher (in the location added to LIBPATH above).

   For OS X, this is libpicmu20d.dylib or libpicmux20d.dylib renamed to libpicmu20.dylib or libpicmux20.dylib and copied over the existing dispatcher (in the location added to DYLD_LIBRARY_PATH above).

3) Edit the pdebug.ini file from your bin folder and modify the line
Logfile=c:\logfile.txt to reflect a valid location in your system to write the log file.

4) Place the edited pdebug.ini file somewhere the dispatcher will find it. The dispatcher will first look for pdebug.ini in /etc/pegasus, but perhaps you can't or won't create the folder, so failing that, the dispatcher will then look for pdebug.ini in the folder pointed to in the SSMPATH environment variable.

5) Run your executable to create the log file named in the pdebug.ini entry for Logfile=

6) Remember to reset the dispatcher library file back to the "release" version when you are finished creating the desired log files.

10.3 Generating AIMTools Debug Log Files on iOS

1) Since iOS Apps are built by statically linking with the dispatcher and opcodes, the App must link with the debug dispatcher. Replace the existing dispatcher (libpici20.a in the location where your app is configured to look for libraries) with libpici20d.a, renamed (this is important) to libpici20.a.

2) Edit the pdebug.ini file from your bin folder and modify the line
Logfile=c:\logfile.txt to contain just the name of the logfile, without a path.

3) Make sure your App is configured to support iTunes file sharing. Using iTunes, copy the pdebug.ini file from your bin folder to the documents folder associated with your App on the device.

4) Run your app to create the log file and retrieve the log file, using iTunes, from the App's document folder.

5) Remember to reset the dispatcher library file back to the "release" version when you are finished creating the desired log files.

10.4 Generating AIMTools Debug Log Files on Android

1) Optionally, have your app add the directory containing the opcodes to its LD_LIBRARY_PATH (see steps 2 and 4 below). In general, this will be the lib directory within your application's directory, e.g., /data/data/com.example.myapp/lib, but it can be determined programmatically from the nativeLibraryDir member of your application's ApplicationInfo, e.g., MyApp.this.getApplicationInfo().nativeLibraryDir.

2) Replace the existing dispatcher (libpicd20.so in the location added to LD_LIBRARY_PATH above) with libpicd20d.so renamed (this is important) to libpicd20.so. Be sure to use the appropriate dispatcher (ARMV5 or ARMV7A) for your target. Alternatively, link your application to the static library dispatcher libpicd20d.a.

3) Edit the pdebug.ini file from your bin folder and modify the line
Logfile=c:\logfile.txt to reflect a valid location in your system to write the log file. If the location is within external storage, make sure that your App has write permissions for external storage, which would be added within your AndroidManifest.xml.

4) Make sure your target is configured to allow developer access. Using ADB place the edited pdebug.ini file somewhere the dispatcher will find it. The dispatcher will first look for pdebug.ini in /mnt/sdcard/pegasus, but perhaps you can't or won't create the folder, so failing that, the dispatcher will then look for pdebug.ini in a folder pointed to in the LD_LIBRARY_PATH environment variable. .

5) Run your app to create the log file and retrieve the log file, using ADB, from the location specified in pdebug.ini.

6) Remember to reset the dispatcher library file back to the "release" version when you are finished creating the desired log files.

11. Deploying PICTools Applications

11.1 Win32 platforms

1) Dispatcher – The PICTools dispatcher must be available to the application that makes calls to Pegasus().
    a. Use PICN20.DLL linked via picnm.lib on MS linkers or picn.lib on non-MS linkers.
    b. Or, preferably, use the static library version of picn20.dll by linking to picn20m.lib (MS linkers), picn20mvs6.lib (MS pre-VS2005) or picn20.lib (other linkers).
2) Opcodes – The PICTools opcodes must be available to the PICTools dispatcher. The PICTools dispatcher will search for and execute the opcodes in standard LoadLibrary path order. First looking for the .SSM version and then the .DLL version of particular opcode library named in the PIC_PARM.Op structure.

11.2 Win64 platforms

1) Dispatcher – The PICTools dispatcher must be available to the application that makes calls to Pegasus().
    a. Use PICX20.DLL linked via picxm.lib.
    b. Or, preferably, use the static library version of picx20.dll by linking to picx20m.lib.
2) Opcodes – The PICTools opcodes must be available to the PICTools dispatcher. The PICTools dispatcher will search for and execute the opcodes in standard LoadLibrary path order. First looking for the .SSM version and then the .DLL version of particular opcode library named in the PIC_PARM.Op structure.
3) Include Files – Make sure the include files from the Win64 development kit, not from the Win32 development kit, are used.
4) Compiler Project Settings – The project used to build the client application must include PIC64 in its list of preprocessor macro settings.

Best and simplest approach in Windows is to place the opcode and dispatcher files in the process executable start up directory. The dispatcher will automatically load the needed opcodes the first time Pegasus is called for the opcode. However, it is also possible to place the opcodes in a known folder and use PegasusLoad() to load them explicitly. Additionally, the opcode SSM files can be embedded in a resource file and loaded using PegasusLoadFromRes(). Additional details on each of these approaches can be found in the PICTools Programmer's Reference.

11.3 Other platforms: Linux, Solaris, AIX, OSX, iOS, Android

Consult the document Deploying PicTools applications on Linux, Solaris (SPARC and x86), AIX and OS X provided in the documentation directory (doc) of the PicTools SDK.   For iOS, consult the "AIMTools iOS QuickStart Guide". For Android, consult the "AIMTools Android QuickStart Guide".

12. Toolkit Software License Agreement

PLEASE READ THE FOLLOWING SOFTWARE LICENSE AGREEMENT WHICH GOVERNS YOUR RIGHT TO USE THE TOOLKIT. YOU MUST ACCEPT THESE TERMS BEFORE YOU ARE ALLOWED TO INSTALL THE TOOLKIT. YOU EXPRESSLY AGREE THAT YOU HAVE THE AUTHORITY TO CONTRACTUALLY BIND THE ORGANIZATION AGREEING TO THESE TERMS.

BY CLICKING "I ACCEPT," OR INSTALLING TOOLKIT, OR PLACING TOOLKIT IN-USE, YOU AGREE TO BE BOUND BY THIS AGREEMENT.

1. GRANT OF LICENSE

a) Evaluation License: This Accusoft Corporation ("ACCUSOFT") Software License Agreement ("AGREEMENT") grants the individual or organization contracting under this agreement ("LICENSEE") a limited, nontransferable, nonexclusive, and non-assignable license to use the trial mode version of this ACCUSOFT Development Toolkit ("TOOLKIT") on a single computer for evaluation of fitness only and not for any commercial purpose; or to use a properly purchased and registered TOOLKIT for development purposes only on a single computer, provided the TOOLKIT is IN-USE on only one computer at any time. (However, additional TOOLKIT licenses may be purchased.) TOOLKIT is "IN-USE" on a computer when it becomes loaded by any means for any purpose into temporary memory (that is, including but not limited to RAM) or when it becomes copied or installed to less temporary storage by any means for any purpose (that is, including but not limited to hard disk, CD-ROM or other removable disk or tape, USB or other flash memory drive or card, or other local, networked, or cloud storage or device) when it is accessible to that computer. The TOOLKIT is explicitly not to be used on a site-wide or on a public or private network basis, via a server or other networked connection.

b) Commercial License: If any of your requirements aren't met by the 1.a. Evaluation License then you need a Commercial License. A Commercial License for a commercial purpose may be acquired by contacting info@accusoft.com. If a separate extended license agreement for TOOLKIT is entered into between Accusoft and LICENSEE at that time, then the terms of that agreement and the Term of that agreement shall govern only where different from the terms and Term of this Agreement. If a separate Accusoft license agreement for TOOLKIT is not entered into at that time, then LICENSEE's permitted use of Program is governed by this Paragraph 1.b., in addition to Paragraph 1.a. Evaluation License, and all other terms and Term are according to this Agreement. In that case, if the Commercial License is an annual license, then the Term of the license grant is for one year beginning on the date of purchase of Program and as may subsequently be extended by Accusoft on LICENSEE's request. If the Commercial License is not an annual license, then the Term of the license grant is until terminated according to this Agreement. In either case, Accusoft grants to LICENSEE a limited, non-exclusive, non-assignable license to install TOOLKIT on a single computer and to use TOOLKIT for a commercial purpose provided the TOOLKIT is IN-USE on only one computer at any time and provided the TOOLKIT is not used on a site-wide basis, or on a public or private network basis, via a server or other networked connection. If a Commercial License is a deployment license for the redistributable runtime portion of TOOLKIT ("PORTION"), then you may reproduce, display, or otherwise distribute and transfer, only as an integral part of a LICENSEE product, the number of RUNTIMES for which a deployment license was provided.

2. EVALUATION LICENSE TELEMETRY DISCLAIMER

By using the TOOLKIT with an Evaluation License, you hereby agree to allow ACCUSOFT to collect and monitor data about which of the TOOLKIT API calls you use and which of our sample programs you use. We use this tracking information to determine which areas of the TOOLKIT are most (and least) popular to improve our SDK capabilities and our samples in the future. The tracking information is anonymized in such a way that neither ACCUSOFT nor anybody else can associate this information with you. Please note that no usage data is collected for a licensed TOOLKIT.

3. REDISTRIBUTION OF TOOLKIT RUNTIMES

ACCUSOFT does not grant LICENSEE any rights to deploy, license, sell, reproduce, copy, install, lease, timeshare, rent, or otherwise distribute or transfer TOOLKIT or any PORTION of TOOLKIT except as provided in Section 1. GRANT OF AGREEMENT as restricted by Section 4. VIDEO TECHNOLOGIES DISTRIBUTION. For licensing information about any other distribution of TOOLKIT or PORTION, please visit our web site (https://www.accusoft.com/licensing.htm), or contact our sales staff. LICENSEE agrees to notify ACCUSOFT immediately of any violations or changes in status regarding LICENSEE's compliance with any term of this AGREEMENT.

In the event that ACCUSOFT grants LICENSEE in a written, separate runtime license agreement ("RUNTIME AGREEMENT") a right to deploy, license, sell, reproduce, copy, install, lease, timeshare, rent, or otherwise distribute or transfer PORTIONS, the RUNTIME AGREEMENT will specify what PORTIONS may be distributed ("RUNTIME"). LICENSEE agrees to acknowledge and uphold the terms and conditions of this AGREEMENT as well as the terms of the RUNTIME AGREEMENT itself, which will be provided only in writing. In such event, LICENSEE may distribute RUNTIMES as part of the LICENSEE's software application or derivative works ("PRODUCT") upon additionally agreeing to the following:

a) LICENSEE understands and acknowledges that in order to receive any discounted pricing for RUNTIME distribution licensing fees based on the type of installation, it must either: 1) prepay for a number of RUNTIME licenses that is sufficient to qualify for ACCUSOFT's then-current published quantity discount, or 2) it must pay for the licenses in accordance with a written contract between LICENSEE and ACCUSOFT.

b) LICENSEE's PRODUCT shall not compete to any degree with the TOOLKIT. Such competitive PRODUCTS are defined as software development toolkits that include similar functionality as TOOLKIT and that are intended for use by software developers and/or system integrators.

c) LICENSEE's PRODUCT must be substantially greater in scope with greater functionality and features than those of the TOOLKIT.

d) LICENSEE will not use ACCUSOFT's name, logo, or trademarks to market PRODUCT without prior written approval of ACCUSOFT except LICENSEE will include a statement substantially similar to the following within PRODUCT documentation and about box: "Portions of this product contain imaging and other technology owned by Accusoft Corporation, Tampa, FL, (www.accusoft.com). ALL RIGHTS RESERVED." See Section 22. THIRD PARTY NOTICES for additional requirements.

e) LICENSEE agrees to only distribute the RUNTIMES. No license or other rights are granted to LICENSEE for any distribution of the TOOLKIT or PORTIONS including, but not limited to, documentation, source code, or the RUNTIME distribution unlock codes.

g) LICENSEE will only distribute the RUNTIMES on the hardware and operating system(s) for which the RUNTIMES are intended to be used according to the RUNTIME AGREEMENT.

If ANY of the terms of this AGREEMENT are not applicable to LICENSEE'S situation, or if any of the terms of this AGREEMENT cannot be complied with, or if LICENSEE needs modifications to this AGREEMENT or the license granted for any reason, LICENSEE must contact ACCUSOFT about obtaining an expanded license from ACCUSOFT (available by phone at: 813-875-7575, by e-mail at: info@accusoft.com or by fax at: 813-875-7705).

This AGREEMENT grants rights to LICENSEE only for the TOOLKIT and does not convey any other rights of any kind including, but not limited to, use or distribution of ACCUSOFT technology.

4. VIDEO TECHNOLOGIES DISTRIBUTION

The TOOLKIT may contain video compression/decompression technologies such as H.265 that are covered by third party patents. If so, ACCUSOFT has obtained patent licenses covering ACCUSOFT's distribution of its TOOLKIT. However ACCUSOFT's licenses do not extend to LICENSEE's PRODUCT. LICENSEE is responsible for obtaining any required third-party patent licenses for distribution of LICENSEE's PRODUCT containing any portions of TOOLKIT that include these technologies. LICENSEE will indemnify and hold ACCUSOFT harmless from any and all patent claims that arise from distribution of LICENSEE's PRODUCT containing any portions of TOOLKIT that include these technologies.

5. OWNERSHIP

LICENSEE acknowledges and agrees that ACCUSOFT owns all rights, title, and interest in the TOOLKIT, in all forms, including without limitation any and all worldwide proprietary rights therein, including but not limited to trademarks, copyrights, patent rights, patent continuations, trade secrets and confidential information.

LICENSEE may not remove or alter the copyright notice from any copy of the TOOLKIT or any copy of the written materials, accompanying the TOOLKIT.

LICENSEE waives its right to contest any of ACCUSOFT's patents, trademarks, service marks, trade names, copyrights, and other intellectual property and proprietary rights in and to the TOOLKIT.

LICENSEE shall not use such trademarks, service marks, and trade names except where and as permitted under this AGREEMENT without receiving ACCUSOFT's prior written approval of such use. If such approval is granted, LICENSEE's right to use such trademarks, service marks, and trade names shall end upon the termination of this AGREEMENT.

6. RESTRICTIONS AND RESERVATIONS

All rights and licenses not expressly granted to LICENSEE are reserved to ACCUSOFT. LICENSEE is strictly prohibited from reproducing, copying, marketing, selling, distributing, licensing, sublicensing, leasing, timesharing, or renting the TOOLKIT or PORTION, and LICENSEE is strictly prohibited from any use of the TOOLKIT or PORTION except as permitted by this AGREEMENT, and such actions are expressly prohibited. LICENSEE is strictly prohibited from incorporating or including the TOOLKIT or PORTION into or as part of any PRODUCT or service of LICENSEE except as provided by this AGREEMENT, regardless of the functionality of TOOLKIT (or lack thereof) within or as part of such PRODUCT or service of LICENSEE. LICENSEE shall not for any reason disassemble, decompile, decrypt, or reverse engineer the TOOLKIT or PORTION or in any manner attempt to discover or reproduce the source code or any other copyrightable, proprietary, or trade secret aspect of the TOOLKIT or PORTION. Nor shall LICENSEE use the TOOLKIT or PORTION, directly or indirectly, in developing LICENSEE's own PRODUCT with, or including, similar functionality. LICENSEE shall not make any copies of the TOOLKIT or PORTION for any purpose whatsoever except as permitted by this AGREEMENT. Source code that is provided with TOOLKIT for sample or demonstration purposes may be used directly or indirectly in developing PRODUCT, however, it may not be distributed in source form in whole or in part with or as part of PRODUCT.

7. WARRANTY DISCLAIMER

LICENSEE ACKNOWLEDGES AND AGREES THAT THE TOOLKIT IS PROVIDED "AS IS." ACCUSOFT DISCLAIMS ANY AND ALL REPRESENTATIONS AND WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND AGAINST INFRINGEMENT.

8. LIMITATION OF LIABILITY

ACCUSOFT SHALL HAVE NO LIABILITY TO LICENSEE, LICENSEE AFFILIATES, SUBSIDIARIES, SHAREHOLDERS, OFFICERS, DIRECTORS, EMPLOYEES, REPRESENTATIVES, OR ANY THIRD PARTY, WHETHER IN CONTRACT, TORT, NEGLIGENCE, OR PRODUCTS LIABILITY, FOR ANY CLAIM, LOSS, OR DAMAGE, INCLUDING BUT NOT LIMITED TO, LOST PROFITS, LOSS OF USE, BUSINESS INTERRUPTION, LOST DATA, LOST FILES, OR FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND OR NATURE WHATSOEVER ARISING OUT OF OR IN CONNECTION WITH USE OF OR INABILITY TO USE THE TOOLKIT, OR THE PERFORMANCE OR OPERATION OF THE TOOLKIT, EVEN IF ACCUSOFT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

9. INDEMNIFICATION BY LICENSEE

LICENSEE SHALL INDEMNIFY, HOLD HARMLESS, AND DEFEND ACCUSOFT FOR ANY LOSS, CLAIM, ACTION, OR PROCEEDING THAT ARISES OR RESULTS FROM ANY ACTIONS OR OMISSIONS OF LICENSEE PERTAINING TO THE PRODUCT OR THE TOOLKIT AND FROM ANY ACTIONS OF LICENSEE THAT ARE IN VIOLATION OF THIS AGREEMENT.

10. TERM AND TERMINATION

Unless otherwise agreed to by the parties, this AGREEMENT shall become effective upon the earlier of LICENSEE's clicking of "I Accept" or LICENSEE'S installing or placing TOOLKIT IN-USE ("Effective Date") and shall continue in full force and effect through Term or until terminated in accordance with the terms set forth in this AGREEMENT.

Any material breach of this AGREEMENT shall automatically and immediately terminate this AGREEMENT. In the event that LICENSEE ceases to do business or is adjudged bankrupt or insolvent, ACCUSOFT may, at its sole option, terminate this AGREEMENT, by giving ten (10) Business Days written notice of such termination, which notice shall identify and describe the basis for such termination.

In the event of any termination of this AGREEMENT, any RUNTIME AGREEMENT is simultaneously terminated and LICENSEE shall stop using the TOOLKIT and PORTION, shall cease manufacturing the PRODUCT containing TOOLKIT or PORTION, and shall cease distributing PRODUCT containing TOOLKIT or PORTION. LICENSEE shall also require its resellers, OEMs, and other distribution channels (if any) to likewise stop manufacturing and distributing the PRODUCT containing TOOLKIT or PORTION. Within ten (10) Business Days thereafter, LICENSEE shall return or, at ACCUSOFT's option, destroy, the TOOLKIT and all PORTIONS, whether or not incorporated in or with the PRODUCT, that are within LICENSEE's possession, custody, and control, and shall certify to ACCUSOFT in writing within ten (10) Business Days after that return or destruction that it has complied with the foregoing obligation.

All sections except Section 1. GRANT OF LICENSE shall continue in full force and effect, notwithstanding any termination of this AGREEMENT.

11. LIQUIDATED DAMAGES

In the event LICENSEE (a) copies the TOOLKIT or PORTION except as permitted by this AGREEMENT, (b) uses the TOOLKIT or PORTION for any reason other than as permitted by this AGREEMENT, (c) installs or uses the TOOLKIT or PORTION on more than a single computer, or (d) otherwise violates or breaches this Agreement, LICENSEE agrees that ACCUSOFT is entitled to obtain as liquidated damages and not as a penalty the greater of the amount of (v) the published quantity one distribution price based upon the type of distribution; (w) $99 per each user of each PRODUCT or service of LICENSEE in which the TOOLKIT or PORTION is included, copied, incorporated, embedded, or accessible; (x) $100 per copy of TOOLKIT or PORTION; (y) $100 per copy of any PRODUCT in which TOOLKIT or PORTION is included, copied, incorporated, embedded, or accessible; or (z) three percent (3%) of all revenues realized by LICENSEE pertaining to any PRODUCTS or services of LICENSEE in which TOOLKIT or PORTION is included, copied, incorporated, embedded, or accessible. THE LICENSEE EXPRESSLY AGREES THAT THE FOREGOING LIQUIDATED DAMAGES ARE NOT A PENALTY.

12. CONFIDENTIALITY

LICENSEE acknowledges that the TOOLKIT contains ACCUSOFT know-how, confidential, and trade secret information ("PROPRIETARY INFORMATION"). LICENSEE agrees: (a) to hold the PROPRIETARY INFORMATION in the strictest confidence, (b) not to, directly or indirectly, copy, reproduce, distribute, manufacture, duplicate, reveal, report, publish, disclose, cause to be disclosed, or otherwise transfer the PROPRIETARY INFORMATION to any third party, (c) not to make use of the PROPRIETARY INFORMATION other than as permitted by this AGREEMENT, and (d) to disclose the PROPRIETARY INFORMATION only to LICENSEE's representatives requiring such material for effective performance of this AGREEMENT and who have undertaken an obligation of confidentiality and limitation of use consistent with this AGREEMENT. This obligation shall continue as long as allowed under applicable law.

13. INJUNCTIVE RELIEF

LICENSEE agrees that any violation or threat of violation of this AGREEMENT will result in irreparable harm to ACCUSOFT for which damages would be an inadequate remedy. Therefore, in addition to its rights and remedies available at law (including but not limited to the recovery of damages for breach of this AGREEMENT), ACCUSOFT shall be entitled to immediate injunctive relief to prevent any violation of ACCUSOFT's copyright, trademark, trade secret rights regarding the TOOLKIT, or to prevent any violation of this AGREEMENT, including, but not limited to, unauthorized use, copying, distribution or disclosure of or regarding the TOOLKIT or PORTION, as well as any other equitable relief as the court may deem proper under the circumstances.

14. NO REDUCED PRICING

In any determination of ACCUSOFT's damages (whether liquidated damages or actual damages), or any determination of any licensing fees or royalties due ACCUSOFT under this AGREEMENT due to a breach by LICENSEE hereunder, LICENSEE shall not be entitled to any discounts (volume or otherwise) or reduced licensing fees or royalties. The foregoing sentence shall be applicable unless LICENSEE has negotiated and entered into a written, signed agreement with ACCUSOFT for such reduced or discounted licensing fees or royalties and paid ACCUSOFT such fees or royalties in advance of any of: (a) distribution of the TOOLKIT or PORTION, (b) copying of the TOOLKIT or PORTION, or (c) incorporation or use of the TOOLKIT or PORTION in or pertaining to any PRODUCT or service of LICENSEE. Further, LICENSEE agrees that it shall not be entitled to reduced licensing fees or royalties when determining ACCUSOFT's damages due to any undertaking or activity by LICENSEE regarding the TOOLKIT or PORTION outside of or exceeding the scope of permission or other terms of this AGREEMENT, or LICENSEE's actions otherwise in violation of this AGREEMENT.

15. ATTORNEYS' FEES AND COSTS

In the event of any lawsuit or other proceeding brought as a result of any actual or alleged breach of this AGREEMENT, to enforce any provisions of this AGREEMENT, or to enforce any intellectual property or other rights in or pertaining to the TOOLKIT or PORTION, the prevailing party shall be entitled to an award of its reasonable attorneys' fees and costs, including the costs of any expert witnesses, incurred at all levels of proceedings.

16. GOVERNING LAW

This AGREEMENT shall be construed, governed, and enforced in accordance with the laws of the State of Florida, without regard to any conflicts of laws rules. Any action related to or arising out of this AGREEMENT will be filed only in the Florida courts and LICENSEE consents to the exclusive jurisdiction and venue of the state and federal courts located in Tampa, Florida.

17. SEVERABILITY

If any provision of this AGREEMENT is determined to be invalid by any court of final jurisdiction, then it shall be omitted and the remainder of the AGREEMENT shall continue to be binding and enforceable. In addition, the Court is hereby authorized to enforce any provision of the AGREEMENT that the Court otherwise deems unenforceable, to whatever lesser extent the Court deems reasonable and appropriate, rather than invalidating the entire provision. Without limiting the generality of the foregoing, LICENSEE expressly agrees that should LICENSEE be found to have breached the AGREEMENT, under no circumstances shall LICENSEE be entitled to any volume or other discount, or reduced licensing fee or royalty in the determination of ACCUSOFT's damages, or otherwise in the determination of any licensing fee or royalty owed to ACCUSOFT.

18. GOVERNMENT RIGHTS

The TOOLKIT and accompanying documentation have been developed at private expense and are sold commercially. They are provided under any U.S. government contracts or subcontracts with the most restricted and the most limited rights permitted by law and regulation. Whenever so permitted, the government and any intermediaries will obtain only those rights specified in ACCUSOFT's standard commercial license. Thus, the TOOLKIT referenced herein, and the documentation provided by Accusoft hereunder, which are provided to any agency of the U.S. Government or U.S. Government contractor or subcontractor at any tier shall be subject to the maximum restrictions on use as permitted by FAR 52.227-19 (June 1987) or DFARS 227.7202-3(a) (Jan. 1, 2000) or successor regulations. Manufacturer is Accusoft Corporation, 4001 N. Riverside Drive Tampa, FL 33603.

19. ENTIRE AGREEMENT

This AGREEMENT represents the entire understanding of the parties concerning the subject matter hereof and supersedes all prior communications and agreements, whether oral or written, relating to the subject matter of this AGREEMENT. Only a writing signed by the parties may modify this AGREEMENT. In the event of any modification in writing of this AGREEMENT, including an expanded license agreement, all sections of this Agreement survive except Section 1. Grant of License.

20. CONTACT US

Should you have any questions concerning this AGREEMENT, or if you desire to contact ACCUSOFT for any reason, please contact ACCUSOFT at 1-813-875-7575.

21. OTHER RESTRICTIONS

a) This AGREEMENT shall not be amended, altered, changed, or modified in any way, unless agreed to in writing by both ACCUSOFT and LICENSEE. Such writing must be executed by a duly authorized representative of ACCUSOFT and a duly authorized representative of LICENSEE.

b) This AGREEMENT is not transferable or assignable by LICENSEE under any circumstances, without the prior written consent of ACCUSOFT. ACCUSOFT will not unreasonably withhold such consent. This AGREEMENT shall be binding upon, and is made for the benefit of, each party, its successors, and permitted assignees (if any). For the purposes of this AGREEMENT, any change in control of LICENSEE shall constitute an assignment or transfer of this AGREEMENT requiring prior written consent of ACCUSOFT. As used in this section, a change in control is defined as (i) any change in ownership of more than fifty percent (50%) of the voting interest in LICENSEE, whether by merger, purchase, foreclosure of a security interest, or other transaction, or (ii) a sale of all or substantially all of the assets of LICENSEE.

c) The relationship established by this AGREEMENT between LICENSEE and ACCUSOFT shall be that of Licensee and Licensor. Nothing contained in this AGREEMENT shall be construed as creating a relationship of agency, joint venture, or partnership between LICENSEE and ACCUSOFT. Neither party shall have any right whatsoever to incur any liabilities or obligations on behalf of the other party.

d) ACCUSOFT's failure to perform any term or condition of this AGREEMENT as a result of conditions beyond its control such as, but not limited to, war, strikes, fires, floods, acts of God, governmental restrictions, power failures, or damage or destruction of any network facilities or servers, shall not be deemed a breach of this AGREEMENT.

e) The headings provided in this AGREEMENT are for convenience and reference purposes only. In the event of a conflict between the terms and conditions listed in this AGREEMENT, and any attached Schedules or Appendices, the terms and conditions of this AGREEMENT shall govern.

f) A waiver of a breach, violation, or default under this AGREEMENT shall not be a waiver of any subsequent breach, violation, or default. Failure of either party to enforce compliance with any term or condition of this AGREEMENT shall not constitute a waiver by the party of such term or condition.

g) All notices and communications shall be in writing and shall be deemed to have been duly given the earlier of when delivered or three (3) Business Days after mailing by certified mail, return receipt requested, postage prepaid, or by international delivery service, addressed to the parties at their respective addresses set forth on the Order Form or at such other addresses as the parties may designate by written notice in accordance with this section.

22. THIRD PARTY NOTICES

a) ADOBE PDF TECHNOLOGY

Portions of the PDF functionality in the TOOLKIT may include Adobe Technology ("Adobe") licensed to ACCUSOFT. If so, the term "TOOLKIT" as defined in the Agreement includes this technology licensed from Adobe and related documentation, and any upgrades, modified versions, updates, additions, and copies thereof.

FONT LICENSE: If the TOOLKIT includes font software, LICENSEE may embed the font software, or outlines of the font software, into its Application to the extent that the font vendor copyright owner allows for such embedding. The fonts contained in this package may contain both Adobe and non-Adobe owned fonts. LICENSEE may fully embed any font owned by Adobe.

LIMITED WARRANTY

THE TERMS OF THIS AGREEMENT STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ACCUSOFT'S BREACH OF WARRANTY. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, ADOBE AND ITS SUPPLIERS MAKE NO WARRANTY, EXPRESS OR IMPLIED, AS TO MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT WILL ADOBE OR ITS SUPPLIERS BE LIABLE TO LICENSEE FOR ANY CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, EVEN IF AN ADOBE REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY THIRD PARTY. Some states or jurisdictions do not allow the exclusion or limitation of incidental, consequential or special damages, or the exclusion of implied warranties, or limitations on how long an implied warranty may last, so the above limitations may not apply to LICENSEE. In such states/countries and to the extent permissible, any implied warranties are limited to thirty (30) days.

INTELLECTUAL PROPERTY.

The Adobe technology is the intellectual property of Adobe and is protected under trademarks, registered trademarks, copyrights, and/or patents in the United States and/or other countries. ALL RIGHTS RESERVED.

b) CAPTIVA ISIS TECHNOLOGY

Any ISIS functionality in the TOOLKIT is licensed to ACCUSOFT from Captiva Software, a division of OpenText Corp. In that case, the term "TOOLKIT" as defined in the Agreement includes technology licensed from Captiva and related documentation, and any upgrades, modified versions, updates, additions, and copies thereof.

LIMITED WARRANTY

THE TERMS OF THIS AGREEMENT STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ACCUSOFT'S BREACH OF WARRANTY. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, CAPTIVA AND ITS SUPPLIERS MAKE NO WARRANTY, EXPRESS OR IMPLIED, AS TO MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT WILL CAPTIVA OR ITS SUPPLIERS BE LIABLE TO LICENSEE FOR ANY CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, EVEN IF A CAPTIVA REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY THIRD PARTY. Some states or jurisdictions do not allow the exclusion or limitation of incidental, consequential or special damages, or the exclusion of implied warranties, or limitations on how long an implied warranty may last, so the above limitations may not apply to LICENSEE. In such states/countries and to the extent permissible, any implied warranties are limited to thirty (30) days.

INTELLECTUAL PROPERTY.

All ISIS technology is the intellectual property of Captiva and is protected under trademarks, registered trademarks, copyrights, and/or patents in the United States and/or other countries. ALL RIGHTS RESERVED.

Rev. 2022-05-13