

# AIMTools™ Programmer's Reference

---

Imaging Development Kit

AIMTools™ version 2.0  
February 15, 2012



## Table of Contents

<b>OVERVIEW .....</b>	<b>1</b>
<b>AIMTOOLS SDK OPCODE MATRIX .....</b>	<b>1</b>
<b>FUNCTIONS .....</b>	<b>3</b>
<b>Pegasus .....</b>	<b>3</b>
<b>PegasusLibInit .....</b>	<b>6</b>
<b>PegasusLibTerm .....</b>	<b>7</b>
<b>PegasusLibThreadInit .....</b>	<b>7</b>
<b>PegasusLibThreadTerm .....</b>	<b>7</b>
<b>PegasusLoad.....</b>	<b>8</b>
<b>PegasusLoadFromRes .....</b>	<b>8</b>
<b>PegasusQuery .....</b>	<b>8</b>
<b>PegasusUnload .....</b>	<b>9</b>
<b>PIC2List Functions .....</b>	<b>10</b>
<b>OPERATIONS .....</b>	<b>25</b>
<b>OP_F2D, OP_F2DPLUS: Convert image file format to DIB.....</b>	<b>25</b>
<b>OP_J2KE, OP_J2KE3D, OP_J2KERGB : Decompress JPEG2000 .....</b>	<b>30</b>
<b>OP_LIE3PLUS: Expand Lossless JPEG and IMStar to DIB or RAW.....</b>	<b>35</b>
<b>OP_S2D: Expand sequential JPEG to DIB or RAW .....</b>	<b>38</b>
<b>OP_ZOOM2: DIB zoom/shrink, gray scale, halftone .....</b>	<b>44</b>
<b>PLUS Opcodes – Beyond 8bpp Grayscale Support.....</b>	<b>45</b>
<b>GENERAL STRUCTURES .....</b>	<b>46</b>
<b>BITMAPINFOHEADER .....</b>	<b>46</b>

PIC_PARM.....	50
QUEUE .....	65
RGBQUAD.....	67
<b>OPERATIONS STRUCTURES.....</b>	<b>68</b>
F2D_STRUC: OP_F2D, OP_F2DPLUS.....	68
LOSSLESS3: OP_LIE3, OP_LIE3PLUS .....	75
J2K_EXTRA: OP_J2KE, OP_J2KE3D, OP_J2KERGB, OP_J2KP, OP_J2KP3D, OP_J2KPRGB.....	81
J2K_UNION: OP_J2KE, OP_J2KE3D, OP_J2KERGB, OP_J2KP, OP_J2KP3D, OP_J2KPRGB.....	82
DIB_OUTPUT: OP_S2D, OP_SE2D,.....	90
ZOOM_PARMS: OP_ZOOM2.....	100
PEGQUERY: PegasusQuery .....	107
REGION .....	112
REGION2 .....	114
<b>APPENDIX 1. CIRCULAR QUEUE PROCESSING .....</b>	<b>117</b>
<b>APPENDIX 2. HANDLING PEGASUS() RESPONSES.....</b>	<b>121</b>



## Overview

The purpose of this reference is to describe the AIMTools API in detail including a detailed description of the functions and data structures provided by the AIMTools API. It is assumed and suggested that you have already read the *PICTools and AIMTools Programmer's Guide*, available with any AIMTools download from [www.accusoft.com](http://www.accusoft.com).

This reference guide has four sections. **Functions** describe the functions provided by the API. **Operations** describe the various image operations supported by the AIMTools API. **General Structures** describe data structures used by AIMTools which have general applicability for all or most operations. **Operation Structures** describes data structures which are used by a small number of operations.

In the following, **opcode** and **operation** are used interchangeably. Strictly, an **opcode** is an operation code, i.e. a manifest numeric constant which is associated with an operation in a one-to-one relationship. An **operation structure** is the operation-specific portion of the generalized AIMTools parameter data structure, the PIC\_PARM data structure. A portion of the PIC\_PARM data structure is a union of a number of **operation structures**. For each operation, only one particular **operation structure** is meaningful in that union. Ordinarily, a reference to a field without any other qualification in the following refers to a field in the PIC\_PARM structure. In the descriptions of **operations** or **operation structures**, a reference to a field without any other qualification may refer to a field in either the **operation structure** or in the PIC\_PARM structure.

In the following, where the term palette is used, it is meant to refer to a color table rather than referring to a palette in the sense of a Windows palette. A color table is an array of RGBQUAD or RGBTRIPLE color values as would be used for a 4-bit or 8-bit DIB. Each pixel on a 4-bit or 8-bit DIB is regarded as an index into the associated color table array. The color value of the pixel is specified by the red, green and blue values in the RGBQUAD structure.

Externally, AIMTools is implemented in a set of libraries. The AIMTools **API library** implements the shell of the AIMTools API. When an AIMTools API function is called for an image operation, the AIMTools API library loads the AIMTools **opcode library** within which the operation is implemented. The AIMTools API library also calls the AIMTools opcode library as necessary to perform the operation. The AIMTools Development Kit provides the following functions, described in the following section:

Function	Description
Pegasus	Access to compression, decompression, transformation and image utility functions
PegasusLoad	Explicitly load an AIMTools opcode library
PegasusLoadFromRes	not applicable
PegasusQuery	Access to information about an image
PegasusUnload	Explicitly unloads an AIMTools opcode library
PegasusLibInit	Used to initialize the library
PegasusLibTerm	Used to terminate the library
PegasusLibThreadInit	Used to initialize the library in each thread
PegasusLibThreadTerm	Used by static lib versions to terminate the library in each thread

## AIMTOOLS SDK Opcode Matrix

The following table relates opcode "numbers" to opcode names and to the actual library represented.

PLEASE NOTE: Toolkits are sold separately for each operating system. The use of AIMTools requires a licensing agreement with Accusoft. This requires a discussion with an Accusoft sales representative about your desired use of AIMTools and the specific features for your application deployment. The AIMTools opcodes remain in evaluation mode until the completion of an agreement with an Accusoft sales representative.

<b>Opcod</b>	<b>Name</b>	<b>iOS File Name</b>	<b>Android File Name</b>
63	OP_LIE3PLUS	libpici6320.a	libpicd6320.so
65	OP_SE2DPLUS	libpici6520.a	libpicd6520.so
69	OP_J2KE	libpici6920.a	libpicd6920.so
85	OP_F2DPLUS	libpici8520.a	libpicd8520.so
88	OP_ZOOM2	libpici8820.a	libpicd8820.so

## Functions

### Pegasus

#### RESPONSE Pegasus(PIC\_PARM PICFAR\* *PicParm*, REQUEST *Request*)

The **Pegasus** function is the single entry point into the image-manipulation operations provided by AIMTools. The specific image operation requested is indicated by the value of the **PicParm->Op** field containing the operation's opcode.

#### Parameters [PicParm](#)

Identifies the operation to perform, specifies additional input and output parameters for the operation and specifies the input and output image buffers (the **Get** and **Put** queues). For additional information, see the description of the [PIC\\_PARM](#) structure in the **AIMTools Structures** section.

#### Request

Specifies the request to perform. It has one of the following values:

Value	Meaning
REQ_CONT	Resumes execution of the operation which was paused when <b>Pegasus</b> returned an operation event (legacy mode only).
REQ_EXEC	Begins execution of the operation. Please note that any error condition or warning Status returned during REQ_INIT may be overwritten during REQ_EXEC and therefore the application should save it if needed, before calling REQ_EXEC.
REQ_INIT	Begins initialization of the operation. Internal memory allocation and other initialization takes place.
REQ_TERM	Terminates the operation. The operation may be terminated at any time. All internally-allocated memory is freed. When <b>Pegasus</b> returns an error (RES_ERR response), the operation is already terminated and REQ_TERM is not needed.

**Return Value** The returned response from **Pegasus** is one of the following values. When a response is only returned for some operations, additional useful information may be found in the description of the operation and of the operation structure.

Value	Meaning
RES_ALLOCATE_APP2_BUF	An APP2 marker was encountered in the image and <b>AppFieldSize</b> was set to -1 in the operation structure. The application should allocate an <b>AppFieldLen</b> size buffer for the APP2 data. This response can occur after REQ_INIT, but does not occur after REQ_EXEC. It may be returned by the OP_S2D and OP_P2D opcodes.
RES_ALLOCATE_COMMENT_BUF	A comment was encountered in the image and <b>PF_AllocateComment</b> was set in the <b>PicFlags</b> field in the operation structure. The

RES_AUX_NEEDED	<p>application should allocate a <b>CommentLen</b> size buffer for the comment. This response can occur after REQ_INIT or after REQ_EXEC or after both. It may be returned by the OP_S2D and OP_P2D opcodes.</p> <p>Auxiliary image data was encountered in the image and it was too large for the value of the <b>AuxSize</b> field in the operation structure. The application should allocate a larger buffer. This response can occur after REQ_INIT or after REQ_EXEC. It is only returned by the OP_F2D and OP_F2DPLUS opcodes.</p>
RES_COLORS_MADE	<p><b>PF_MakeColors</b> was set in <b>PicFlags</b> in the operation structure and an optimum color table has been made. The application can now make use of the color table. For example, a palette based on the color table can be created and realized before DIB pixels based on the color table are returned.. This response only occurs after REQ_EXEC. It is only returned by the OP_S2D and OP_P2D group of opcodes.</p>
RES_DONE	<p>The operation was completed without error. This response occurs after REQ_INIT and after REQ_EXEC.</p>
RES_ERR	<p>An error occurred. Examine <b>Status</b> in the PIC_PARM data for an error code. The operation is terminated.</p>
RES_EXTEND_PIC2LIST	<p>Comment or other data has been encountered in the input image and <b>PIC2ListSize</b> is not 0 and <b>PIC2ListSize - PIC2ListLen</b> indicates there is not enough space in the <b>PIC2List</b> buffer to hold all of the data. When RES_EXTEND_PIC2LIST is returned, <b>PIC2ListLen</b> has been set to the new <b>PIC2ListSize</b> required for the <b>PIC2List</b> buffer in order to hold all of the encountered data. <b>PacketType</b> has the PIC2List packet type for the data so the application can decide whether or not it cares. Ordinarily, the application will reallocate <b>PIC2List</b> to be <b>PIC2ListLen</b> bytes long and will set <b>PIC2ListSize</b> equal to <b>PIC2ListLen</b> before returning from DeferFn.</p>
RES_GET_DATA_YIELD	<p><b>PF_YieldGet</b> was specified in <b>PicFlags</b> in the operation structure and an additional portion of the <b>Get</b> queue was consumed. This response does not indicate the <b>Get</b> queue is empty. The application may perform any other desired processing. This response can occur after REQ_INIT and after REQ_EXEC.</p>
RES_GET_NEED_DATA	<p>The <b>Get</b> queue has not been allocated or doesn't contain enough data to continue the operation. The application must allocate a buffer for the <b>Get</b> queue, if necessary, and must add data to the <b>Get</b> queue before resuming the operation. This response can occur after REQ_INIT and after REQ_EXEC.</p>

RES_HAVE_COMMENT	<p>A comment was encountered in the image data and <b>PF_AllocateComment</b> was set in <b>PicFlags</b> in the operation structure and the PIC_PARM <b>CommentLen</b> field is not 0. The fact that <b>CommentLen</b> is not 0 indicates that this comment is not the first comment in the image data. The application should save the previous comment's data before it is overwritten by the new comment. After the operation is resumed, <b>Pegasus</b> will immediately return a RES_ALLOCATE_COMMENT_BUF response to allow the application to allocate a buffer for the new comment's data. This response can occur after REQ_INIT or after REQ_EXEC or after both. It may be returned by the OP_S2D and OP_P2D opcodes.</p>
RES_NULL_PICPARM_PTR	<p>A NULL PIC_PARM pointer was passed to the Pegasus routine. This indicates an error in programming.</p>
RES_POKE	<p>Data needs to be poked into a prior location in the output image, and that location is no longer present in the <b>Put</b> queue. This will only occur after all other, sequential, output has been placed in the <b>Put</b> queue. <b>SeekInfo</b> has the offset in the output image to poke the data. The data extends sequentially (no wrap-around) from <b>Put.FrontEnd</b>. The length of the data is <b>Put.RearEnd - Put.FrontEnd</b>. RES_POKE is intended to be optional and the output image is still correct. RES_POKE's are intended to occur after all other sequential or RES_SEEK'ed output has occurred. RES_POKE is a combination of seek/write in one response/operation and the write is ordinarily a small number of bytes, like a DWORD.</p>
RES_PUT_DATA_YIELD	<p><b>PF_YieldPut</b> was specified in <b>PicFlags</b> in the operation structure and some additional data was placed in the <b>Put</b> queue. This response does not indicate the <b>Put</b> queue is full. The application should perform any desired processing. This response can occur after REQ_INIT and after REQ_EXEC. It is recommended that you do not change the PUT queue pointers in response to this message.</p>
RES_PUT_NEED_SPACE	<p>There is additional output data available, but the <b>Put</b> queue isn't allocated or doesn't have enough room for the additional output data. The application must allocate a buffer for the <b>Put</b> queue, if necessary, or remove data from the <b>Put</b> queue before resuming the operation. This response can occur after REQ_INIT and after REQ_EXEC. Note that RES_PUT_NEED_SPACE is not returned after the last operation output data is placed in the</p>

RES_SEEK	<p>queue. You should check the <b>Put</b> queue after the operation is complete to see if additional output data has been produced.</p> <p>The <b>Get</b> queue or <b>Put</b> queue must be repositioned to a non-contiguous location in the input or output image. This response can only occur after REQ_EXEC. It may be returned by the OP_F2D, OP_F2DPLUS, OP_D2F, OP_D2FPLUS or OP_UTL opcodes.</p> <p>RES_SEEK is an indication ordinarily that substantial sequential output from the new location is expected. Ordinarily RES_SEEK in the Put queue would occur when reading an existing image through the Put queue for a multi-image append.</p>
RES_YIELD	<p>This response is not currently returned by any opcodes.</p>
RES_QUERY	<p>NA for AIMTools. This response is used by the ScanFix opcode for each object found during certain object detection operations. The calling function provides a structure with information about the object in question, and will keep or remove the object based on the data returned in the structure.</p>

**Errors** If the RES\_ERR response is returned, examine **Status** in the PIC\_PARM structure to determine the error value. See the ERRORS.H file for descriptive names for each error code, except for -1. When an error code of -1 is returned, a low-level **Pegasus** routine has detected an error which was almost certainly caused by invalid image data. It can be most correctly treated as identical to ERR\_BAD\_DATA.

---

## PegasusLibInit

### BOOL PegasusLibInit (DWORD *hInstance*)

API call to initialize the PIC libraries. This is never strictly required as Pegasus() will initialize on the first call by an application.

**Parameters** **hInstance**

Specifies the module instance handle for the application or 0 on non-Windows platforms

**Return Value** The return value is TRUE if able to initialize the library.

**Notes** **PegasusLibTerm** may then be called after the completion of all **Pegasus** library functions to terminate the library in the unusual case that all resources must be freed before the application process terminates.

---

## PegasusLibTerm

### PegasusLibTerm ()

API call to terminate the PIC libraries. This is only needed to terminate the library in the unusual case that all resources must be freed before the application process terminates.

**Parameters** none

**Return Value** none

**Notes** This call frees the memory used by **PegasusLibInit** or **Pegasus()**

---

## PegasusLibThreadInit

### BOOL PegasusLibThreadInit ()

API call to initialize the PIC libraries in a thread. This will be called from within the new thread. **PegasusLibInit** will have been called previously in the application.

**Parameters** none

**Return Value** none

**Notes** **PegasusLibThreadInit** must be called before any other **Pegasus** calls in a multi-threaded application. **PegasusLibThreadTerm** must then be called from the same thread after the completion of all **Pegasus** library functions and before the thread terminates.

---

## PegasusLibThreadTerm

### PegasusLibThreadTerm ()

API call to terminate the PIC libraries for a thread. This is only needed in a multi-threaded application. This must be called from within the same thread that called **PegasusLibThreadInit()** before the thread terminates.

**Parameters** none

**Return Value** none

**Notes** This call is required to free the memory used by **PegasusLibThreadInit**

---

## PegasusLoad

**LONG PegasusLoad(OPERATION *Op*, LONG *ParmVer*, char PICFAR\* *Path*)**

The **PegasusLoad** function explicitly loads a specific AIMTools opcode library.

**Parameters**    **Op**

Specifies the opcode library to be loaded.

**ParmVer**

Specifies the version of the library to load, ordinarily CURRENT\_PARMVER.

**Path**

Specifies the directory location from which to load the library. If the application specifies 0 for the path, then the opcode library is explicitly loaded using the same directory search order as when it is implicitly loaded (see **Notes**).

**Return Value**    The return value is ERR\_NONE if able to load the requested library, otherwise an error value defined in ERRORS.H is returned.

**Notes**            An AIMTools opcode library is ordinarily implicitly loaded the first time **Pegasus** is called for the opcode and is unloaded when the application terminates.

Using **PegasusLoad** allows an application a way to directly specify the directory containing the opcode library. It also allows the opcode library to be loaded at a time when error-handling may be more convenient, in case the library can't be loaded. An application is not required to call the **PegasusUnload** function after calling **PegasusLoad**, nor is an application required to call **PegasusLoad** before calling **PegasusUnload**.

---

## PegasusLoadFromRes

Note: this function is not applicable in AIMTools.

---

## PegasusQuery

**BOOL PegasusQuery(PIC\_PARM PICFAR\* *PicParm*)**

The **PegasusQuery** function examines an image to determine image properties such as image size, dimensions and format.

**Parameters**    [PicParm](#)

Specifies the image properties to be returned and specifies the input buffer containing at least the image header. The requested properties are also returned in **PicParm**. For additional information, see the description of the [PIC\\_PARM structure](#) in the **General Structures** section. Also see the description of the [PEGQUERY structure](#) in the **Operation Structures** section.

**Return Value**    The return value is TRUE if the function successfully determines the requested image properties. The image properties are returned in **PicParm**.

- Errors** If FALSE is returned, first examine the **PicParm->Status** value. If **Status** is non-zero, it holds the error code. If **Status** is 0, then some of the requested image properties could not be returned. Examine the **u.QRY.BitFlagsAck** field to determine which image properties were returned.
- Notes** Before calling **PegasusQuery**, the application zeros **PicParm** and initializes its **ParmSize**, **ParmVer** and **ParmVerMinor** fields. If an image comment is desired, then the **Comment** and **CommentSize** fields are initialized to a buffer for the comment. The application allocates a buffer and initializes the **Get** queue fields. At least the image header must have been loaded into the buffer. **PegasusQuery** will treat this buffer as the start of the image file and will parse information from the image header and the image data if possible. The application must also have set one or more flags in **u.QRY.BitFlagsReq** indicating which image properties should be returned.
- PegasusQuery** returns TRUE or FALSE to indicate whether it could return all of the requested image properties. The bit flags in **u.QRY.BitFlagsAck** are set to indicate which image properties were returned. Information about the image is returned in the **Head**, **ColorTable**, **Comment**, **CommentLen** and **u.QRY** areas. **PegasusQuery** may return image properties which were not requested by the **BitFlagsReq** field.
- PegasusQuery** cannot be called using PIC\_PARM data which is currently in use by another Pegasus function (i.e., if **PicParm->Reserved** is not 0).
- Note that **PegasusQuery** alters the contents of the PIC\_PARM structure. Some care should therefore be taken when using a **PegasusQuery** PIC\_PARM in a subsequent call to **Pegasus**. Specifically, since **PegasusQuery** changes the **QRY** data in the PIC\_PARM operation-specific union, and since the **Pegasus** operation will use a different operation structure for accessing that same union, you will ordinarily zero **QRY** after retrieving the important image properties and before initializing the **Pegasus** operation structure.

---

## PegasusUnload

**PUBLIC VOID PegasusUnload (OPERATION Op, LONG ParmVer)**

The **PegasusUnload** function explicitly unloads one or more of the AIMTools' opcode libraries.

**Parameters** **Op**

Specifies the opcode library to be unloaded.

**ParmVer**

Specifies the version of the library to unload.

**Return Value** This function does not return a value.

**Notes** Ordinarily the AIMTools opcode libraries are implicitly unloaded when the application terminates.

Using **PegasusUnload** allows the application a method to explicitly unload a specific opcode library if desired. The use of **PegasusUnload** is always optional. **PegasusUnload** does not have to be called if **PegasusLoad** is called and **PegasusUnload** can be called if **PegasusLoad** is not called.

The AIMTools API library maintains a load count for each opcode library. **PegasusUnload** must be called as many times as **PegasusLoad** was called in

order for the library to be unloaded. If the opcode library were implicitly loaded by calling **Pegasus** without calling **PegasusLoad**, then the opcode library will be unloaded the first time **PegasusUnload** is called, regardless of how many times **Pegasus** may have been called.

It is not an error to call **PegasusUnload** while a Pegasus operation for the opcode is active. The opcode library will be unloaded when the Pegasus operation is complete or terminated.

---

## PIC2List Functions

The PIC2List functions allow the AIMTools PIC2List data to be searched, stored and retrieved. PIC2List provides AIMTools operations with a way of returning information to an application when the amount of information and the types of information are not known. It also provides an application with a way of providing this information to AIMTools operations. Ordinarily, the information is not strictly necessary to the AIMTools operation. For example, if an image comment is not returned to the client application, the image can still be decompressed. Similarly, if the client application doesn't supply an image comment, an image can still be compressed.

A particular item of information, such as a single comment, is stored in a PIC2List packet. Different types of information are stored in different kinds of PIC2List packets. See also the AIMTools include file `pic2file.h` for additional information. The PIC2List Functions are implemented in the `pic2list.c` file with header `pic2list.h`. These files are included in the AIMTools SDK `samples\common` folder.

### PIC2List General Functions

#### P2LValid

```
BOOL P2LValid(const P2LIST* p2l);
```

P2LValid returns TRUE if the PIC2List appears to be valid. If `p2l->list` is 0, then `p2l->len` and `p2l->size` must be 0. Otherwise, `p2l->size` must be greater than or equal to `p2l->len` and the list is walked from beginning to end to make sure that the packet lengths and the binary 0 byte following the last packet are consistent with `p2l->len`.

#### P2LAlloc

```
BOOL P2LAlloc(P2LIST* p2l, DWORD size);
```

P2LAlloc allocates a new PIC2List or enlarges a previously-allocated PIC2List. P2LAlloc returns TRUE if the allocation succeeds, otherwise a memory allocation error has occurred and P2LAlloc returns FALSE.

#### P2LFree

```
void P2LFree(P2LIST* p2l);
```

P2LFree frees the PIC2List's buffer and sets `p2l->list`, `p2l->len` and `p2l->size` to 0.

### PIC2List Generic Packet Functions

The P2PktGeneric structure is defined as:

```
{
    BYTE Type;
    DWORD Length;
    BYTE Data[1];
}
```

```
    } P2PktGeneric;
```

**P2LInsert**

```
P2PktGeneric* P2LInsert(P2LIST* p2l, P2PktGeneric*
pktInsertBefore, BYTE type, DWORD len, BYTE* data)
```

P2LInsert inserts a PIC2List packet of type with data of length len immediately before an existing packet pktInsertBefore. If pktInsertBefore, insert at end. If list is empty, ignore pktInsertBefore and just add the packet. If data == 0, then the packet is allocated and the data is set to 0. Returns 0 if a memory allocation error occurs and on an attempt to add a P2P\_EOF packet

**P2LAdd**

```
P2PktGeneric* P2LAdd(P2LIST* p, BYTE type, DWORD len, const
BYTE* data);
```

P2LAdd adds a packet to a PIC2List, enlarging the list if necessary. The packet type is type, the packet data, of length len bytes, is pointed to by data. If data is a null pointer the packet is added to the list, but the packet data area isn't initialized. A pointer is returned to the added packet. If a null pointer is returned, then a memory allocation error occurred attempting to enlarge the list.

**P2LDelete**

```
BOOL P2LDelete(P2LIST* p2l, P2PktGeneric* pkt);
```

P2LDelete deletes a packet from the PIC2List. pkt must point into the PIC2List to the Type field of the packet to be deleted. No validation is performed.

**P2LFirst**

```
P2PktGeneric* P2LFirst(const P2LIST* p2l);
```

P2LFirst returns a pointer to the first packet in the PIC2List. P2LFirst returns 0 if the list is empty or if the only packet in the list is the PIC2List EOF packet.

**P2LNext**

```
P2PktGeneric* P2LNext(const P2LIST* p2l, P2PktGeneric*
pkt);
```

P2LNext returns a pointer to the packet following a given packet. Pkt must point into the PIC2List to the Type field of a packet. No validation is performed. If 0 is returned, then there are no packets following the packet.

**P2LFind**

```
P2PktGeneric* P2LFind(const P2LIST* p2l, BYTE type);
```

P2LFind returns a pointer to the first packet of the requested type. P2LFind returns 0 if there is no packet in the PIC2List of the requested type.

**P2LFindNext**

```
P2PktGeneric* P2LFindNext(const P2LIST* p2l, const
P2PktGeneric* pkt);
```

P2LFindNext returns a pointer to the next packet following pkt whose type is pkt->Type. pkt must point into the PIC2List to the Type field of a packet. No validation is performed. P2LFindNext returns 0 if there are no packets following pkt in PIC2List whose type is the same as pkt->Type.

**PIC2List Registration Packet Functions**

The P2PktRegistration structure is defined as:

```

{
    BYTE   Type;
    DWORD  Length;
    LONG   Op;
    DWORD  Code;
    char   Name[1];
} P2PktRegistration;

```

**P2LAddOrModifyRegistration**

```

P2PktRegistration* P2LAddOrModifyRegistration(
    P2LIST*      p2l,
    OPERATION    Op,
    DWORD        Code,
    const char*  pszName);

```

P2LAddOrModifyRegistration adds a registration packet to the PIC2List. If a packet for opcode Op already exists, then it is modified, otherwise a new packet for opcode Op is added. A registration packet is added to PIC2List to avoid an AIMTools evaluation dialog for a registered AIMTools operation occurring within a different AIMTools operation.

**P2LDeleteRegistration**

```

BOOL P2LDeleteRegistration(P2LIST* p2l, P2PktRegistration*
pkt);

```

P2LDeleteRegistration is called to delete a registration packet from the PIC2List. pkt must point to the Type field of a registration packet within the PIC2List. No validation is performed.

**P2LFindRegistration**

```

P2PktRegistration* P2LFindRegistration(P2LIST *p2l,
OPERATION Op);

```

P2LFindRegistration returns a pointer to the registration packet for opcode Op. P2LfindRegistration returns 0 if there is no registration packet for opcode Op.

**PIC2List RawData Packet Functions**

The P2PktRawData structure is defined as:

```

{
    BYTE   Type;
    DWORD  Length;
    CHAR   RawDescription[4];
    DWORD  RawLength;
    BYTE   RawData[1];
} P2PktRawData;

```

**P2LAddRawData**

```

P2PktRawData* P2LAddRawData(
    P2LIST*      p2l,
    const char *desc,
    DWORD        len,
    const BYTE*  data);

```

P2LAddRawData adds a raw data packet to the PIC2List, enlarging the list if necessary. It is not an error to have more than one raw data packet in a PIC2List with the same desc. If data is a null pointer, then the RawData field in the added raw data packet is not initialized. P2LAddRawData returns a pointer to the new packet. If P2LAddRawData returns 0, then a memory allocation error occurred attempting to enlarge the list for the new packet.

#### **P2LDeleteRawData**

```
BOOL P2LDeleteRawData(P2LIST* p2l, P2PktRawData* pkt);
```

P2LDeleteRawData is called to delete a raw data packet from the PIC2List. pkt must point to the Type field of a raw data packet within the PIC2List. No validation is performed.

#### **P2LFindRawData**

```
P2PktRawData* P2LFindRawData(const P2LIST* p2l, const char* desc);
```

P2LFindRawData finds the first raw data packet in the PIC2List with a RawDescription matching desc. P2LFindRawData returns 0 if there are no raw data packets with a matching RawDescription.

#### **P2LFirstRawData**

```
P2PktRawData* P2LFirstRawData(const P2LIST* p2l);
```

P2LFirstRawData returns a pointer to the first raw data packet with any RawDescription in the PIC2List. P2LFirstRawData returns 0 if there are no raw data packets.

#### **P2LNextRawData**

```
P2PktRawData* P2LNextRawData(const P2LIST* p2l, P2PktRawData* pkt);
```

P2LNextRawData returns a pointer to the first raw data packet following pkt. pkt must point into the PIC2List to the Type field of a raw data packet. No validation is performed. If 0 is returned, then there is no raw data packet following the packet.

#### **P2LFindNextRawData**

```
P2PktRawData* P2LFindNextRawData(const P2LIST* p2l, const P2PktRawData* pkt);
```

P2LFindNextRawData returns a pointer to the raw data packet following pkt whose RawDescription matches pkt->RawDescription. Pkt must point into the PIC2List to the Type field of a raw data packet. No validation is performed. P2LFindNextRawData returns 0 if there are no raw data packets following pkt in the PIC2List with a matching pkt->RawDescription.

#### **PIC2List TIFF / EXIF Packet Functions**

The P2PktTiffTag structure is defined as:

```
{
    BYTE  Type;
    DWORD Length;
    BYTE  Location;
    WORD  TiffTag;
    WORD  TiffType;
    DWORD TiffCount;
    BYTE  TiffData[1];
} P2PktTiffTag;
```

**P2LAddTiffTag**

```

P2PktTiffTag* P2LAddTiffTag(
    P2LIST*    p2l,
    BYTE      loc,
    WORD      tag,
    WORD      type,
    DWORD     count,
    const BYTE* data);

```

P2LAddTiffTag adds an Exif tag packet of the specified characteristics, enlarging the PIC2List if necessary. loc specifies the IFD within which the Exif tag packet should be placed. If data is a null pointer, then the TiffData field in the Exif tag packet is uninitialized. P2LAddTiffTag returns 0 if an Exif tag packet with the same tag and the same IFD is already present in the PIC2List (see P2LModifyTiffTag and P2LAddOrModifyTiffTag). P2LAddTiffTag returns 0 if a memory allocation error occurs attempting to enlarge the PIC2List.

**P2LModifyTiffTag**

```

P2PktTiffTag* P2LModifyTiffTag(
    P2LIST*    p2l,
    BYTE      loc,
    WORD      tag,
    WORD      type,
    DWORD     count,
    const BYTE* data);

```

P2LModifyTiffTag modifies the Exif tag packet in the loc IFD with the specified type, enlarging the PIC2List if necessary. If data is 0, then the TiffData field in the Exif tag packet is uninitialized. P2LModifyTiffTag returns 0 if no Exif tag packet with the same tag and the same IFD is already present (see P2LAddTiffTag and P2LAddOrModifyTiffTag). P2LModifyTiffTag returns 0 if a memory allocation error occurs attempting to enlarge the PIC2List.

**P2LAddOrModifyTiffTag**

```

P2PktTiffTag* P2LAddOrModifyTiffTag(
    P2LIST*    p2l,
    BYTE      loc,
    WORD      tag,
    WORD      type,
    DWORD     count,
    const BYTE* data);

```

P2LAddOrModifyTiffTag adds an Exif tag packet of the specified characteristics, enlarging the PIC2List if necessary. loc specifies the IFD within which the Exif tag packet should be placed. If data is a null pointer, then the TiffData field in the Exif tag packet is uninitialized. If an Exif tag packet with the same tag and the same IFD is already present in the PIC2List, it is modified, otherwise a tag is added to the PIC2List. P2LAddOrModifyTiffTag returns 0 if a memory allocation error occurs attempting to enlarge the PIC2List.

**P2LDeleteTiffTag**

```

BOOL P2LDeleteTiffTag(P2LIST* p2l, P2PktTiffTag* pkt);

```

P2LDeleteTiffTag is called to delete an Exif tag packet from the PIC2List. pkt must point to the Type field of an Exif tag packet within the PIC2List. No validation is performed.

**P2LFirstTiffTag**

**P2PktTiffTag\*** P2LFirstTiffTag(const P2LIST\* p2l, BYTE loc);

P2LFirstTiffTag returns a pointer to the first Exif tag packet of any type in the loc IFD. P2LFirstTiffTag returns 0 if there are no tiff tag packets in the loc IFD.

**P2LNextTiffTag**

**P2PktTiffTag\*** P2LNextTiffTag(const P2LIST\* p2l, const P2PktTiffTag\* pkt);

P2LNextTiffTag returns a pointer to the first Exif tag packet following pkt in the same IFD as pkt. pkt must point into the PIC2List to the Type field of an Exif tag packet. No validation is performed. If 0 is returned, then there is no Exif tag packet following the packet in the same IFD.

**P2LFindTiffTag**

**P2PktTiffTag\*** P2LFindTiffTag(const P2LIST\* p2l, BYTE loc, WORD tag);

P2LFindTiffTag finds the first Exif tag packet in the loc IFD with a matching tag. P2LFindTiffTag returns 0 if there is no matching Exif tag packet in the IFD.

**P2LFindNextTiffTag**

**P2PktTiffTag\*** P2LFindNextTiffTag(const P2LIST\* p2l, const P2PktTiffTag\* pkt);

P2LFindNextTiffTag returns a pointer to the Exif tag packet following pkt in the same IFD. Pkt must point into the PIC2List to the Type field of an Exif tag packet. No validation is performed. P2LFindTiffTag returns 0 if there are no Exif tag packets following pkt in the same IFD.

**P2LTiffNthData**

**DWORD** P2LTiffNthData(const P2PktTiffTag\* pkt, DWORD n);

P2LTiffNthData retrieves the Nth data item in an Exif tag. The BYTE, SHORT, LONG, SBYTE, SSHORT or SLONG data item is returned as a DWORD,

**P2LTiffCount**

**DWORD** P2LTiffCount(P2PktTiffTag\* pkt);

P2LtiffCount return the count of data items in an Exif tag.

**P2LTiffType**

**WORD** P2LTiffType(P2PktTiffTag\* pkt);

P2LTiffType returns the data type for an Exif tag.

**P2LTiffTag**

**WORD** P2LTiffTag(P2PktTiffTag\* pkt);

P2LTiffTag returns the tag for an Exif tag.

**P2LTiffLocation**

**BYTE** P2LTiffLocation(P2PktTiffTag\* pkt);

P2LTiffLocation returns the IFD location for the Exif tag.

**P2LTiffData**

**DWORD** P2LTiffData(P2PktTiffTag\* pkt);

P2LTiffData returns the first data item in an Exif tag. The BYTE, SHORT, LONG, SBYTE, SSHORT or SLONG data item is returned as a DWORD,

**P2LTiffByte**

**BYTE P2LTiffByte (P2PktTiffTag\* pkt);**

P2LTiffByte returns the BYTE data in an Exif tag.

**P2LTiffShort**

**BYTE P2LTiffShort (P2PktTiffTag\* pkt);**

P2LTiffShort returns the WORD data in an Exif tag.

**P2LTiffLong**

**BYTE P2LTiffLong (P2PktTiffTag\* pkt);**

P2LTiffLong returns the DWORD data in an Exif tag.

**P2LTiffRationalNumerator**

**BYTE P2LTiffRationalNumerator (P2PktTiffTag\* pkt);**

P2LTiffRationalNumerator returns the numerator for an Exif tag rational data type as a DWORD.

**P2LTiffRationalDenominator**

**BYTE P2LTiffRationalDenominator (P2PktTiffTag\* pkt);**

P2LTiffRationalDenominator returns the denominator for an Exif tag rational data type as a DWORD.

**P2LTiffAscii**

**LPCSTR P2LTiffAscii (P2PktTiffTag\* pkt);**

P2LTiffAscii returns a pointer to the ASCIIZ data in an Exif tag whose data type is ASCII.

**P2LTiffUndefined**

**BYTE\* P2LTiffUndefined (P2PktTiffTag\* pkt);**

P2LTiffUndefined returns a pointer to the BYTE array data in an Exif tag whose data type is UNDEFINED.

**PIC2List MO:DCA Packet Functions**

The P2PktModca structure is defined as:

```
{
    BYTE   Type;
    DWORD  Length;
    DWORD  SFID;
    DWORD  SFLen;
    BYTE   SFData[1];
} P2PktModca;
```

**P2LAddModca**

**P2PktModca\* P2LAddModca (P2LIST\* p2l, DWORD ID, DWORD Len, BYTE\* data)**

Add a modca PIC2List packet with SFID ID. If data == 0, then the packet is allocated and the data is set to 0. Returns 0 if a memory allocation error occurs.

**P2LDeleteModca**

**P2LDeleteModca (p2l, pkt)**

P2LDeleteModca deletes a packet from the PIC2List. pkt must point into the PIC2List to the Type field of the packet to be deleted. No validation is performed.

**P2LFirstModca****P2LFirstModca** (p21)

P2LFirstModca returns a pointer to the first Modca packet in the PIC2List.  
 P2LFirstModca returns 0 if the list is empty or if the only packet in the list is the PIC2List EOF packet.

**PIC2List IOCA Packet Functions**

The P2PktIoca structure is defined as:

```

{
    BYTE  Type;
    DWORD Length;
    WORD  SDFID;
    WORD  SDFLen;
    BYTE  SDFData[1];
} P2PktIoca;

```

**P2LAddIoca**

**P2PktIoca\*** P2LAddIoca (P2LIST\* p21, DWORD ID, DWORD Len, BYTE\* data)

Add a Ioca PIC2List packet with SFID ID. If data == 0, then the packet is allocated and the data is set to 0. Returns 0 if a memory allocation error occurs.

**P2LFindIoca**

**P2PktIoca\*** P2LFindIoca (P2LIST\* p21, DWORD ID)

P2LFindIoca returns a pointer to the first Ioca packet of type ID in the PIC2List.

**P2LFindNextIoca**

**P2PktIoca\*** P2LFindNextIoca (P2LIST\* p21, P2PktIoca\* pkt)

P2LFindIoca returns a pointer to the next Ioca packet of type ID and following pkt.

**PIC2List CALS Packet Functions**

The P2PktCals structure is defined as:

```

{
    BYTE  Type;
    DWORD Length;
    char  CalsID[9];
    char  CalsData[1];
} P2PktCals;

```

**P2LAddCals**

**P2PktCals\*** P2LAddCals (P2LIST\* p21 const char\* ID, DWORD Len, const char\* data)

Add a Cals PIC2List packet with SFID ID. If data == 0, then the packet is allocated and the data is set to 0. Returns 0 if a memory allocation error occurs.

**P2LFindCals**

**P2PktCals\*** P2LFindCals (const P2LIST\* p21, const char\* ID)

P2LFindCals returns a pointer to the first Cals packet of type ID in the PIC2List.

**P2LFindNextCals**

```
P2PktCals* P2LFindNextCals (const P2LIST* p2l, const
P2PktCals* pkt)
```

P2LFindNextCals returns a pointer to the next Cals packet of type ID and following pkt.

**PIC2List PNGText Packet Functions**

The P2PktPngText structure is defined as:

```
{
    BYTE   Type;
    DWORD  Length;
    char   Key[80]; // including null terminator
    char   Text[1];
} P2PktPngText;
```

**P2LAddPngText**

```
P2PktPngText* P2LAddPngText(P2LIST* p2l, const char* Key,
DWORD Len, const char* Text)
```

Add a PngText PIC2List packet with Key and Text. Key is null-terminated and Text has length Len and does not have to be null-terminated. If Text is 0, then space is allocated for Text and 0'ed zero'ed..

**P2LFindPngText**

```
P2PktPngText* P2LFindPngText (const P2LIST* p2l, const
char* Key)
```

P2LFindPngText returns a pointer to the first PngText packet packet of key Keyt.

**P2LFindNextPngText**

```
P2PktPngText* P2LFindNextPngText (const P2LIST* p2l, const
P2PktPngText* pkt)
```

P2LFindNextPngText a pointer to the next PngText PIC2List packet with the same key as and following pkt.

**PIC2List JBIG2 Comment Packet Functions**

The P2PktJBIG2Comment structure is defined as:

```
{
    BYTE   Type;
    DWORD  Length;
    DWORD  PageNum;
    DWORD  ValueOfs;
    BYTE   CharSet;
    char   NameThenValue[1];
} P2PktJBIG2Comment;
```

**P2LAddJBIG2Comment**

```
P2PktJBIG2Comment* P2LAddJBIG2Comment(P2LIST* p2l, DWORD
PageNum, DWORD SegNum, BYTE CharSet, DWORD NumNameBytes,
const BYTE* Name, DWORD NumValueBytes, const BYTE* Value)
```

Add a JBIG2Comment PIC2List packet with Name and Value. Name has length NumNameBytes bytes. Value has length NumValueBytes bytes.If

NumNameBytes is zero, then Name must be null-terminated and its length will be determined.

If NumValueBytes is zero, then Value must be null-terminated and its length will be determined. If Name is zero and NumNameBytes is nonzero, then space is allocated and zero-filled. If Value is zero and NumValueBytes is nonzero, then space is allocated and zero-filled.

CharSet is either P2JBIG2Com\_8bitchars (ANSI, ISO/IEC 8859) or P2JBIG2Com\_16bitchars (UNICODE, ISO/IEC 10646 UCS-2)

#### **P2LFindJBIG2Comment**

```
P2PktJBIG2Comment* P2LFindJBIG2Comment (const P2LIST* p2l,
    DWORD PageNum, DWORD SegNum)
```

P2LFindJBIG2Comment returns a pointer to the first JBIG2Comment for page PageNum.

#### **P2LFindNextJBIG2Comment**

```
P2PktJBIG2Comment* P2LFindNextJBIG2Comment (const P2LIST*
    p2l, const P2PktJBIG2Comment* pkt)
```

P2LFindNextJBIG2Comment returns a pointer to the next JBIG2 comment for page PageNum and following pkt.

### **PIC2List IPTCDataSet Packet Functions**

The P2PktIPTCDataSet structure is defined as:

```
{
    BYTE   Type;
    DWORD  Length;
    BYTE   Record;
    BYTE   DataSet;
    DWORD  DataSetType;
    DWORD  DataSetLength;
    BYTE   DataField[1];
} P2PktIPTCDataSet;
```

Type	Set to P2P_IPTCDataSet to indicate that this packet contains an IPTC DataSet
Length	The total length of this packet in bytes.
Record	The IPTC Record number for this DataSet (1..9)
DataSet	The IPTC DataSet number for this DataSet
DataSetType	IPTC_UNDEFINED, IPTC_BYTE, IPTC_ASCII, IPTC_INTEGER, IPTC_3_10_Struct, IPTC_3_110_Struct
DataSetLength	The Length, in bytes, of the following DataField field
DataField.	The DataSet's DataField

#### **P2LAddIPTCDataSet**

```
P2PktIPTCDataSet* P2LAddIPTCDataSet (
    P2LIST*   p2l,
    BYTE      rec,
    BYTE      set,
    DWORD     type,
    DWORD     length,
    Const BYTE* data )
```

P2LAddIPTCDataSet adds an IPTC DataSet packet of the specified characteristics, enlarging the PIC2List if necessary. If the type is IPTC\_UNDEFINED then it will be set according to the standard. If data is a null pointer, then the DataField field in the IPTC DataSet is set to 0. P2LAddIPTCDataSet will return a 0 if a memory allocation error occurs while attempting to enlarge the PIC2List.

#### **P2LDeleteIPTCDataSet**

```
BOOL P2LDeleteIPTCDataSet(P2LIST* p2l, P2PktIPTCDataSet* pkt)
```

P2LDeleteIPTCDataSet is called to delete an IPTC DataSet packet from the PIC2List. Pkt must point to the Type field of an IPTCDataSet packet within the PIC2List. No validation is performed.

#### **P2LFindIPTCDataSet**

```
P2PktIPTCDataSet* P2LFindIPTCDataSet(const P2LIST* p2l, BYTE rec, BYTE set);
```

P2LFindIPTCDataSet finds the first IPTC DataSet packet in the list with Record equal to rec and DataSet equal to set. P2LFindIPTCDataSet will return a 0 if there is no matching IPTC DataSet packet in the PIC2List.

#### **P2LFirstIPTCDataSet**

```
P2PktIPTCDataSet* P2LFirstIPTCDataSet(const P2LIST* p2l);
```

P2LFirstIPTCDataSet will return a pointer to the first IPTC DataSet packet of any type. P2LFirstIPTCDataSet will return a 0 if there are no IPTC DataSet packets in the PIC2List.

#### **P2LModifyIPTCDataSet**

```
P2PktIPTCDataSet* P2LModifyIPTCDataSet(const P2LIST* p2l, const P2PktIPTCDataSet* pkt, BYTE rec, BYTE set, DWORD type, DWORD length, const BYTE* data);
```

P2LModifyIPTCDataSet modifies the IPTC DataSet pointed to by pkt, enlarging the PIC2List if necessary. Pkt must point to the Type field of an IPTCDataSet packet within the PIC2List. If the type is IPTC\_UNDEFINED then it will be set according to the standard. If data is 0, then the DataField field in the IPTC DataSet packet is uninitialized. P2LModifyIPTCDataSet will return a 0 if pkt is not in the PIC2List. P2LModifyIPTCDataSet will return a 0 if a memory allocation error occurs attempting to enlarge the PIC2List.

#### **P2LNextIPTCDataSet**

```
P2PktIPTCDataSet* P2LNextIPTCDataSet(const P2LIST * p2l, const P2PktIPTCDataSet* pkt);
```

P2LNextIPTCDataSet will return a pointer to the first IPTC DataSet packet following pkt. Pkt must point into the PIC2List to the Type field of an IPTC DataSet packet. No validation is performed. If 0 is returned, then there is no IPTC DataSet packet following the packet in the PIC2List.

**PIC2List JPEG2000 Metadata Packets Functions**

The JPEG2000 Metadata Packets include packets for handling UUID Boxes, XML Boxes, UUID Info Boxes and IPTC Data. The IPTC packets are discussed in their own section.

The P2PktUUIDBox struct is defined as

```
{
    BYTE   Type;
    DWORD  Length;
    BYTE   UUID[16];
    DWORD  DataLength;
    BYTE   Data[1];
} P2PktUUIDBox;
```

Type	Set to P2P_UUIDBox to indicate that this packet contains a UUID Box
Length	The total length of this packet in bytes.
UUID	The 16 byte UUID for this box.
DataLength	The length of the Data field in bytes. A DataLength of zero indicates an empty UUID box.
Data	The data for the UUID box.

**P2LAddUUIDBox**

```
P2PktUUIDBox *P2LAddUUIDBox(P2LIST* p2l, BYTE *UUID, DWORD
length, const BYTE* data);
```

P2LAddUUIDBox adds a UUIDBox packet of the specified characteristics, enlarging the PIC2List if necessary. If Data is a null pointer, then the Data field in the UUID Box is set to 0. If length is zero then an empty UUID box is added. P2LAddUUIDBox will return a 0 if a memory allocation error occurs while attempting to enlarge the PIC2List.

**P2LDeleteUUIDBox**

```
BOOL P2LDeleteUUIDBox(P2LIST* p2l, P2PktUUIDBox* pkt);
```

P2LDeleteUUIDBox is called to delete a UUID Box packet from the PIC2List. Pkt must point to the Type field of a P2PktUUIDBox packet within the PIC2List. No validation is performed.

**P2LFirstUUIDBox**

```
P2PktUUIDBox *P2LFirstUUIDBox(const P2LIST* p2l);
```

P2LFirstUUIDBox will return a pointer to the first UUID Box packet. P2LFirstUUIDBox will return a 0 if there are no UUID Box packets in the PIC2List.

**P2LNextUUIDBox**

```
P2PktUUIDBox *P2LNextUUIDBox(const P2LIST* p2l,
P2PktUUIDBox* pkt);
```

P2LNextUUIDBox will return a pointer to the first UUID Box packet following pkt. Pkt must point into the PIC2List to the Type field of an UUID Box packet. No validation is performed. If 0 is returned, then there is no UUID Box packet following the packet in the PIC2List.

### P2LFindUUIDBox

```
P2PktUUIDBox* P2LFindUUIDBox(const P2LIST* p2l, BYTE
*UUID) ;
```

P2LFindUUIDBox finds the first UUID Box packet in the list with pkt->UUID equal to UUID. P2LFindUUIDBox will return a 0 if there is no matching UUID Box packet in the PIC2List.

### P2LFindNextUUIDBox

```
P2PktUUIDBox* P2LFindNextUUIDBox(const P2LIST* p2l,
P2PktUUIDBox* pkt) ;
```

P2LFindNexUUIDBox will return a pointer to the first UUID Box packet following pkt with matching UUID of pkt->UUID. Pkt must point into the PIC2List to the Type field of an UUID Box packet. No validation is performed. If 0 is returned, then there is no matching UUID Box packet following the packet in the PIC2List.

The P2PktUUIDDDataEntryURL is defined as

```
{
    BYTE   Type;
    DWORD  Length;
    BYTE   ID;           // Reserved, Must always be zero
    BYTE   Vers;        // always 0
    DWORD  Flag;        // always 0
    BYTE   UUID[16];
    char   URL[1]; // Always NULL terminated
} P2PktUUIDDDataEntryURL;
```

Type	Set to P2P_UUIDDDataEntryURL to indicate that this packet contains a UUID Data Entry URL
Length	The total length of this packet in bytes.
ID	Reserved, must always be zero.
Vers	The Vers field from the UUID Data Entry URL box must always be zero. It will always be zero when returned from an opcode.
Flag	The Flag field from the UUID Data Entry URL box must always be zero. It will always be zero when returned from an opcode.
UUID	The 16 byte UUID for this Data Entry URL.
URL	A NULL terminated UTF-8 string.

### P2LAddUUIDDDataEntryURL

```
P2PktUUIDDDataEntryURL *P2LAddUUIDDDataEntryURL(P2LIST* p2l,
BYTE *UUID, BYTE ID, BYTE Vers,
DWORD Flag, const char* URL);
```

P2LAddUUIDDDataEntryURL adds a UUIDDDataEntryURL packet of the specified characteristics, enlarging the PIC2List if necessary. ID, Vers, and Flag are all reserved and should be set to 0. P2LAddUUIDBox will return a 0 if a memory allocation error occurs while attempting to enlarge the PIC2List.

### **P2LDeleteUUIDDDataEntryURL**

```
BOOL P2LDeleteUUIDDDataEntryURL(P2LIST* p2l,  
P2PktUUIDDDataEntryURL * pkt);
```

P2LDeleteUUIDDDataEntryURL is called to delete a UUID Data Entry URL packet from the PIC2List. Pkt must point to the Type field of a P2PktUUIDDDataEntryURL packet within the PIC2List. No validation is performed.

### **P2LFirstUUIDDDataEntryURL**

```
P2PktUUIDDDataEntryURL *P2LFirstUUIDDDataEntryURL(const  
P2LIST* p2l);
```

P2LFirstUUIDDDataEntryURL will return a pointer to the first UUID Data Entry URL packet. P2LFirstUUIDDDataEntryURL will return a 0 if there are no UUID Data Entry URL packets in the PIC2List.

### **P2LNextUUIDDDataEntryURL**

```
P2PktUUIDDDataEntryURL *P2LNextUUIDDDataEntryURL(const  
P2LIST* p2l, P2PktUUIDDDataEntryURL * pkt);
```

P2LNextUUIDDDataEntryURL will return a pointer to the first UUID Data Entry URL packet following pkt. Pkt must point into the PIC2List to the Type field of an UUID Data Entry URL packet. No validation is performed. If 0 is returned, then there is no UUID Data Entry URL packet following the packet in the PIC2List.

### **P2LFindUUIDDDataEntryURL**

```
P2PktUUIDDDataEntryURL * P2LFindUUIDDDataEntryURL(const  
P2LIST* p2l, BYTE *UUID, const char *URL, int method);
```

P2LFindUUIDDDataEntryURL finds the first UUID Data Entry URL in the list based on method. If method is P2FIND\_UUIDDATAENTRYURL\_BY\_UUID then it will return the first packet in the list with pkt->UUID equal to UUID. If method is P2FIND\_UUIDDATAENTRYURL\_BY\_URL then it will return the first packet in the list with pkt->URL equal to URL. P2LFindUUIDDDataEntryURL will return a 0 if there is no matching UUID Data Entry URL packet in the PIC2List.

### **P2LFindNextUUIDDDataEntryURL**

```
P2PktUUIDDDataEntryURL * P2LFindNextUUIDDDataEntryURL(const  
P2LIST* p2l, P2PktUUIDDDataEntryURL * pkt, int method);
```

P2LFindNexUUIDDDataEntryURL will return a pointer to the next UUID Data Entry URL in the list based on method. If method is P2FIND\_UUIDDATAENTRYURL\_BY\_UUID then it will return the next packet in the list with the UUID at pkt->UUID equal to the UUID at ptr. If method is P2FIND\_UUIDDATAENTRYURL\_BY\_URL then it will return the next packet in

the list with the URL at pkt->URL equal to the URL at ptr. Pkt must point into the PIC2List to the Type field of an UUID Data Entry URL packet. No validation is performed. If 0 is returned, then there is no matching UUID Data Entry URL packet following the packet in the PIC2List.

The P2PktXMLBox struct is defined as

```
{
    BYTE   Type;
    DWORD  Length;
    char   XML[1]; // Always NULL terminated
} P2PktXMLBox;
```

Type	Set to P2P_XMLBox to indicate that this packet contains an XML Box.
Length	The total length of this packet in bytes.
XML	The XML data for this XML Box. It must always be NULL terminated. If the data is UTF-16 it must start with a Byte Order Marker (BOM) and be doubly NULL terminated.

### P2LAddXMLBox

```
P2PktXMLBox *P2LAddXMLBox(P2LIST* p2l, const char* XML);
```

P2LAddXMLBox adds a XML Box packet of the specified characteristics, enlarging the PIC2List if necessary. The XML data must be NULL terminated. If it is UTF-16 data then it must begin with a Byte Order Marker (BOM) and then be terminated by two NULLs. If it is UTF-8 data then it must be terminated with a single NULL. P2LAddXMLBox will return a 0 if a memory allocation error occurs while attempting to enlarge the PIC2List.

### P2LDeleteXMLBox

```
BOOL P2LDeleteXMLBox(P2LIST* p2l, P2PktXMLBox* pkt);
```

P2LDeleteXMLBox is called to delete a XML Box packet from the PIC2List. Pkt must point to the Type field of a P2PktXMLBox packet within the PIC2List. No validation is performed.

### P2LFirstXMLBox

```
P2PktXMLBox *P2LFirstXMLBox(const P2LIST* p2l);
```

P2LFirstXMLBox will return a pointer to the first XML Box packet. P2LFirstXMLBox will return a 0 if there are no XML Box packets in the PIC2List.

### P2LNextXMLBox

```
P2PktXMLBox *P2LNextXMLBox(const P2LIST* p2l, P2PktXMLBox* pkt);
```

P2LNextXMLBox will return a pointer to the first XML Box packet following pkt. Pkt must point into the PIC2List to the Type field of an XML Box packet. No validation is performed. If 0 is returned, then there is no XML Box packet following the packet in the PIC2List.

## Operations

---

### OP\_F2D, OP\_F2DPLUS: Convert image file format to DIB

See the [F2D\\_STRUC: OP\\_F2D, OP\\_F2DPLUS](#) section for additional information. Supported input image file formats are Bitmap, PCX, TIFF, Raw Fax and Targa. **OP\_F2DPLUS** also supports GIF and TIFF/LZW formats.

#### General Notes

- **F\_RelaxedQueue** should be always OR'ed into **PicParm.Flags** for **OP\_F2D**. When the flag is set, the behavior of its **Get** and **Put** queues is the same as for all other opcodes except for **OP\_UTL**.
- The image data output to the **Put** queue is either a DIB or a BMP file image (starting after the 14-byte BITMAPFILEHEADER). If the output is to be a BMP file image including the BITMAPINFOHEADER and color table, then **PF\_IncludeBMPHead** must be set in **F2D.PicFlags**. Whether or not **PF\_IncludeBMPHead** is set, **F2D.BiOut** is initialized to the BITMAPINFOHEADER data describing the output image and **ColorTable** is initialized to the output image color table, if any.
- Each line of pixels in the output image will be padded to a DWORD boundary if needed.
- If the input image, and therefore the output image, is a gray scale image -- that is if the input image has a color table and if all the colors in the color table are shades of gray -- then when **OP\_F2D** is complete, **PF\_IsGray** will be set in **F2D.PicFlags**.
- **F2D.AuxSpace** may be set to point to a block of memory whose size is **F2D.AuxSize**. Data may be returned by **OP\_F2D** in this block of memory organized as a sequential series of one or more "chunks". Each chunk consists of a DWORD **Tag**, followed by a DWORD **Length**, followed by **Length** bytes of data. Thus the total length of each chunk is **Length + 8**. Whenever **OP\_F2D** places data in **F2D.AuxSpace**, it sets **AuxUsed** to the total length consumed in **F2D.AuxSpace**. **D2F.AuxSpace** is currently not used for PCX input files. See the following sections for additional information specific to an input file format.

#### Deprecated Queue Behavior

- If **F\_RelaxedQueue** is not set in **PicParm.Flags**, **OP\_F2D**'s **Get** and **Put** queue behavior requires some additional considerations described below. This deprecated behavior has been retained to provide backwards compatibility to existing applications that expect it when using the opcode. New applications should set **F\_RelaxedQueue** which allows using **OP\_F2D** in a manner consistent with all the other opcodes except for **OP\_UTL**.
- If **F\_RelaxedQueue** is not set in **PicParm.Flags** and the **Get** queue is large enough for the entire input image, the flag **F2D.AllocType** should be set to 1. In this case, with **F2D.AllocType** set to 1, any input queue RES\_SEEK response can be handled as:

```

case RES_SEEK:
    if ( ( PicParm.SeekInfo & SEEK_FILE ) == 0 )
    {
        PicParm.Get.Front =
            PicParm.Get.Start +
            ( PicParm.SeekInfo & SEEK_OFFSET );
    }
    else
    {
        /* if the Put queue is also large enough to hold
           the entire output image */
        PicParm.Put.Front = PicParm.Put.Rear =
            PicParm.Put.Start +
            ( PicParm.SeekInfo & SEEK_OFFSET );
    }
    break;

```

- Also if **F2D.AllocType** is 1, the **Get** queue size must be, or must appear to be, at least one byte larger than the size of the image. Since the extra byte will not be accessed by **OP\_F2D**, this might be coded as:

```

PP.Get.Start = DibPointer;
PP.Get.End   = PicParm.Get.Start + DibLength + 1;
PP.Get.Front = PicParm.Get.Start;
PP.Get.Rear  = PicParm.Get.End - 1;

```

**Get.Front** will be advanced by **OP\_F2D** as it examines the input image data to create the output image. However the image data will not actually have been consumed by **OP\_F2D** so the data between **Get.Start** and **Get.Front** must not be modified.

Similarly, the **Put** queue size must be at least one byte larger than the size of the image. For example:

```

PP.Put.Start = (BYTE *) malloc(OutSize+1);
PP.Put.End   = PP.Put.Start + OutSize + 1;
PP.Put.Front = PP.Put.Rear = PP.Put.Start;

```

## BMP Input

Whether or not **PF\_IncludeBMPHead** is set, the input BMP supplied to the **Get** queue must include the BITMAPINFOHEADER and color table when applicable. Older-format BMP files with a BITMAPCOREHEADER instead of a BITMAPINFOHEADER are also supported. The only field in **Head** which must be initialized according to the input image is **Head.biCompression**. No input image color table need be in **ColorTable**.

The input image may be 1, 4, 8, 16, 24 or 32 bits-per-pixel. The output BMP is 24 bits-per-pixel for 16 and 32 bits-per-pixel input images. Otherwise the output BMP bit depth is the same as the input image bit depth.

4 or 8-bit input images may be uncompressed or RLE compressed. Other input image bit depths must be uncompressed.

If **F2D.AuxSpace** is not 0 and **F2D.AuxSize** is not 0, then any extra data present in the input BMP following the color table and preceding the image DIB bits is returned as an **AUX\_BMP\_EXTRABI** chunk. See the [OP\\_D2F BMP Output](#) section for additional discussion about this data.

If this **AUX\_BMP\_EXTRABI** data is exactly 24 bytes of zeroes, then the input BMP is an OS/2 2.1-compatible bitmap file.

Otherwise, the **AUX\_BMP\_EXTRABI** data could have been used to embed an image comment or image annotation into the input image.

## PCX Input

The input image may be 1, 4, 8 or 24 bits-per-pixel. The output BMP bit depth is the same as the input image bit depth. **D2F.AuxSpace** is ignored for PCX input.

**OP\_F2D** supports version 3.0 PCX files (PCX image header version field has the value 5).

## Targa Input

The following Targa input image types are supported:

- Type 1: uncompressed 1, 4 or 8 bits-per-pixel palletized
- Type 2: uncompressed 1, 4 or 8 bits-per-pixel gray scale
- Type 3: uncompressed 16, 24 or 32 bits-per-pixel RGB color
- Type 9: RLE-compressed 1, 4 or 8 bits-per-pixel palletized
- Type 10: RLE-compressed 1, 4 or 8 bits-per-pixel gray scale
- Type 11: RLE-compressed 16, 24 or 32 bits-per-pixel RGB color

The output BMP bit depth is the same as the input image bit depth.

**OP\_F2D** supports the Targa version 2.0 specification.

If **F2D.AuxSpace** is not 0 and **F2D.AuxSize** is not 0, then data from the Image ID, the Developer Area and the Extension Area are output as chunks to the **F2D.AuxSpace** buffer. If an Image ID is present, it is output with a tag of **AUX\_TGA\_COMMENT**. If Developer Fields are present in the Targa image, then the total number of Developer Fields is output with a tag of **AUX\_TGA\_DEVELOPER\_COUNT**. The Developer Fields themselves are each output with a tag of **AUX\_TGA\_DEVELOPER**. If an Extension Area is present, then 495 bytes are output with a tag of **AUX\_TGA\_EXTENSION**. The first 490 bytes contain the first 490 bytes of the Extension Area. The last 5 bytes are output as 0.

## Raw Fax

Set **Head.biCompression** to **BI\_G3** or **BI\_G32D** to convert raw fax input images to DIB format. Up to 32768 pixels per scan line are supported. The output DIB is 1 bit-per-pixel. The DIB will be upside down compared to the normal DIB orientation (thus the top screen line will be the first line in the DIB).

**F2D.BiOut.biHeight** and **F2D.BiOut.biSizeImage** will not be valid until all the input has been processed. If **PF\_IncludeBMPHead** is set in **F2D.BiOut.PicFlags**, then the **BITMAPINFOHEADER biWidth**, **biHeight** and **biSizeImage** fields will be output as -1. Set **RawG3FillOrder** to the TIFF FillOrder corresponding to the input raw fax image. For a raw fax, this value will ordinarily be 2 (or 0) for a FillOrder filling bytes from the least-significant-bit to the most-significant-bit. Otherwise set **RawG3FillOrder** to 1 for a FillOrder filling bytes from the most-significant-bit to the least-significant-bit. Set **RawG3PhotometricInterpretation** to the TIFF PhotometricInterpretation corresponding to the input raw fax image. For a raw fax, this value will ordinarily

be 0 (WhitelsZero). Otherwise set **RawG3PhotometricInterpretation** to 1 (BlackIsZero).

## TIFF

The input TIFF file may contain multiple images. The **ImageNumber** field in PIC\_PARM is the number of the image which is to be returned as a DIB. 0 and 1 both return the first image – the image whose Image File Directory (IFD) is pointed to by the TIFF header. 2 returns the image whose IFD is pointed to by the first image's IFD. And so on.

The input image may be a PhotometricInterpretation type 0, 1, 2, 3, or 5 TIFF image. Supported Compression types include 1 (uncompressed), 2 (modified G3), 3 (G3), 4 (G4) or 32773 (Packbits). **OP\_F2DPLUS** additionally supports Compression type 5 (LZW), 6 and 7 (JPEG), 8 or 32946 (ZLIB). If the compression type is 6 or 7 (JPEG) then PhotometricInterpretation type 2 (RGB) or 6 (YCBCR) is allowed. **OP\_F2DPLUS** also supports Differencing Predictor types 1 (no predictor) and 2 (horizontal differencing predictor) for Compression type 5. Intel and Motorola byte ordering are supported for all types. The output DIB bits per pixel are the same as the input TIFF samples-per-pixel times the TIFF bits-per-sample.

The Base TIFF version 6.0 specification is supported. Supported extensions include CCIT Bilevel encoding only. Tiled TIFF images are not supported. Planar TIFF images are not supported. Data sample format specifications are not supported. TIFF images with a PhotometricInterpretation=5 (CMYK) will set the F2D.BiOut.Compression field to BI\_CMYK, however no color conversion to RGB is performed. YCBCR and CIELAB images are not supported. Extra pixel components such as opacity or alpha data are not supported.

If **F2D.AuxSpace** is not 0 and **F2D.AuxSize** is not 0, then TIFF image data specified by TIFF tags which are not used in converting to a DIB are output as chunks to the **F2D.AuxSpace** buffer. The chunk tag is AUX\_TIFF\_Base + the value of the TIFF tag.

The image compression method is returned as **F2D.Compression**. The value is the same as the TIFF compression type as above, except that 2-dimensional G3 is returned as -3.

## GIF

**OP\_F2DPLUS** converts GIF87a and GIF89a images to DIB format. The output DIB is always 8-bits-per-pixel. If the GIF input file contains multiple images, only the first image is converted. Interlaced GIF is supported. Transparent GIF is supported. The output DIB is always 8 bits-per-pixel.

Set **PF\_ApplyTransparency** in **PIC\_PARM.u.F2D.PicFlags** to honor a GIF transparent color index in the input file. If **PF\_ApplyTransparency** is set, then the **Put** queue must be large enough for the entire image. The **Put** queue must already have been initialized with the image data onto which the input GIF image is being overlaid, although **Put.Front** and **Put.Rear** must be set to **Put.Start** as usual, denoting an empty queue. The GIF image pixels whose color index is the same as the transparent color index will not be set in the **Put** queue, allowing the previous image data to show through those GIF image pixels.

If **PF\_ApplyTransparency** is not set, and the GIF image has a transparent color index, then the DIB output pixels corresponding to the transparent GIF pixels are set to the color whose color table index is the same as the transparent color index. Any previous image data does not show through those GIF image pixels.

If **PIC\_PARM.Head.biCompression** is BI\_GIFI, then the input image must be an interlaced GIF image. If **biCompression** is BI\_GIFN, then the input image must not be an interlaced GIF image. If **biCompression** is BI\_GIFu, then the input image may be either interlaced or not interlaced.

If the input image is interlaced, then **PIC\_PARM.u.F2D.ProgressiveMode** may be used to specify that the image will be displayed progressively, or that only the complete image will be displayed (see table below). If the **ProgressiveMode** indicates that the image will be displayed progressively, then Pegasus will return RES\_DONE from Pegasus(REQ\_EXEC) each time the data for the next interlace pass is complete. When RES\_DONE is returned after the final interlace pass, **PF\_EOIFound** will be set in **PIC\_PARM.u.F2D.PicFlags**. **PF\_EOIFound** will be clear when RES\_DONE is returned after any other interlace pass.

When RES\_DONE is returned after an interlace pass, you will ordinarily display the image data for that interlace pass from the **Put** queue. You will then empty the **Put** queue by setting **Put.Front** and **Put.Rear** to **Put.Start**. To be useful, progressive modes 1 and 4 assume that the data from the prior interlace pass remain in the **Put** queue. If **PF\_EOIFound** is not set, you will finally call Pegasus(REQ\_CONT) for the image data from the next interlace pass.

<b>ProgressiveMode</b>	<b>Meaning</b>
0	RES_DONE is returned four times. The first DIB returned is 1/8 the height of the full image. The second DIB returned is 1/4 the height of the full image. The third DIB returned is 1/2 the height of the full image. The final DIB returned is the complete image at full size.
1	RES_DONE is returned four times. Each returned DIB is the same size as the complete, full-size image. The first DIB has every 8 <sup>th</sup> line, the second DIB has every 4 <sup>th</sup> line and the third DIB has every 2 <sup>nd</sup> line. In each case, the lines between the interlace pass lines are filled with copies of the preceding interlace pass line. The final DIB is the complete image at full size.
2	RES_DONE is returned once. The returned DIB is the complete image at full size.
3	RES_DONE is returned four times. Each returned DIB is the same size as the complete, full-size image. The first DIB has every 8 <sup>th</sup> line, the second DIB has every 4 <sup>th</sup> line, the third DIB has every 2 <sup>nd</sup> line and the fourth DIB is the complete image. The first DIB fills in the lines between each 8 <sup>th</sup> line with copies of the preceding line. The second DIB adds image data from the second interlace pass to the first DIB. The third DIB adds image data from the third interlace pass to the second DIB. The final DIB adds image data from the fourth interlace pass to the third DIB. Only the first DIB fills in the lines between the current interface pass lines.

If **F2D.AuxSpace** is not 0 and **F2D.AuxSize** is not 0, then extension GIF image data are returned if present. The AUX data includes the entire GIF extension data are returned beginning with the extension introducer '!'. The AUX tag AUX\_GIF\_1 is used for the graphics control extension. AUX\_GIF\_2 is used for the plain text extension. AUX\_GIF\_3 is used for application extensions. AUX\_GIF\_4 is used for comment extensions. Only extensions which precede the first image are returned in a multi-image GIF file.

---

## OP\_J2KE, OP\_J2KE3D, OP\_J2KERGB : Decompress JPEG2000

See the [J2K\\_UNION](#), [TILECOMP](#), [TILE](#), [PARTITION](#), [REGION](#), [REGION2](#) sections for additional information.

### General Notes

JPEG2000 decompression may be used to decompress images with varying grayscale and color bit-depths. Decompression options include thumbnail sizes, resolution detail, and progressive stages.

- **OP\_J2KE** requires that **ParmVerMinor** be no less than 3.
- The Get queue must be at least 1 byte (however a rewind RES\_SEEK will occur if the Get queue is less than 12 bytes).
- The Put queue must be at **u.J2K.StripSize** bytes in size.

The input JPEG2000 compressed image can be in the JPEG2000 raw codestream format or in the JPEG2000 file format. The former is an excerpt of the latter.

The output decompressed image can be formatted as a Windows DIB with each row padded to a 32-bit boundary or as raw image data formatted as specified by the **Region** parameters. The decompression process performs no color reduction or other reformatting of the compressed data. Partitions are decompressed to separate **Put** buffers for reconstruction by the application.

The decompression process follows the familiar REQ\_INIT – REQ\_EXEC–REQ\_TERM sequence of calls to Pegasus. Before REQ\_INIT, some image data is placed in a **Get** buffer and a PIC\_PARM structure is allocated and the basic structure members are initialized (**PicParm.ParmSize** = sizeof(PIC\_PARM), **PicParm.ParmVer** = CURRENT\_PARMVER, **PicParm.ParmVerMinor** = 3, **PicParm.Op** = OP\_J2KE). It is recommended that a **DeferFn** function be defined and F\_UseDeferFn be OR'ed into **PicParm.Flags**. REQ\_INIT will return the minimum size for the **Put** buffer in **u.J2K.StripSize**. REQ\_INIT will also return image information in the **u.J2K.Region** structure.

In order to support larger images, this opcode supports the optional use of **REGION2**. If **ParmVerMinor** is set to 4, then the opcode treats **u.J2K.Region** as a **REGION2** structure instead of a **REGION** structure. Otherwise, **ParmVerMinor** should be set to 3. See the **REGION2** section for more information.

For even larger images or other situations where memory is a concern, this opcode offers a reduced-memory mode of operation. More details of this mode can be found in the Reduced-Memory Mode section below.

The application allocates a **Put** buffer according to **StripSize** and the REQ\_EXEC phase fills the **Put** buffer with decompressed image data.

If **PF\_YieldPut** is set in **u.J2K.PicFlags** then **OP\_J2KE** will return a RES\_PUT\_DATA\_YIELD response periodically during decompression. This would allow an application to progressively display the image being expanded, or to update expansion progress reporting.

OP\_J2KE returns image file comments using **PIC2List** P2P\_Comment packets. JPEG 2000 errors or warnings returned from the underlying JPEG 2000 library optionally return additional descriptive text in **PIC2List** if there is enough space

or if the application reacts to RES\_EXTEND\_PIC2LIST responses. In that case, ERR\_PIC2LIST\_HAS\_DESCRIPTION is returned in **Status** and the information about the error appears in P2PktErrorText packets with **Type** P2P\_ErrorText. See include\pic2file.h and the **PIC2List** section of **Accessing Comments and Other Auxiliary Data** in the PICTools and AIMTools Programmer's Guide for more information about **PIC2List** packets.

## RGB Images

If an image has:

- at least three components,
- none of the first three components are signed, subsampled or color-mapped, and
- the first three components are each 8 bits per sample

then the image is decompressed as a 24-bit interleaved RGB image. If the file is a JP2 file format, then the color space must be RGB and the first three components must not be remapped, i.e., they must be ordered red, green and then blue in the compressed file.

If `F_Bmp` is set in **PicParm.Flags**, then the output image rows are padded to a 32-bit (DWORD) boundary, the output image is bottom up and the output interleaved RGB pixels' samples are ordered blue, green, and then red. Otherwise, set `F_Raw` in **PicParm.Flags**, and set `RF_SwapRB` and `RF_TopDown` in **u.J2K.Region.Flags** if desired. Set `RF_SwapRB` before `REQ_EXEC`, and `OP_J2KE` decompresses to interleaved pixels with samples ordered red, green, and then blue. Otherwise the decompressed interleaved pixels' samples are ordered blue, green, and then red.

If `F_Raw` is set, set **Region.Stride** if desired after `RES_DONE` is returned from `REQ_INIT` and before calling `REQ_EXEC`. If set, be sure to adjust **StripSize** to correspond and make sure the **Put** queue size is at least as big as the adjusted **StripSize** as:

```
StripSize = ( StripSize / Region.Stride ) * <NewStride>;
Region.Stride = <NewStride>;
```

## Color-mapped Images

If an image's first component:

- is color-mapped,
- the sample bit depth is no more than 8 bits,
- the samples aren't signed,
- the color map has no more than 256 colors, and
- the mapped colors are 8-bits per color sample,

then the image can be decompressed as a color-mapped image. Packed pixel output (more than one pixel per byte) is not supported. `OP_J2KE` sets **PicParm.Head.biClrUsed** to the number of colors in the color map and sets **PicParm.ColorTable** according to the colors in the color map as read from the palette in the JPEG2000 image. **PicParm.ColorTable** is available after `REQ_INIT` for the application to use for reserving space in the output file but may change during `REQ_EXEC` to satisfy the setting of `RF_SwapRB` in

**u.J2K.Region.Flags** evaluated during REQ\_EXEC, in which case the application should rewrite the **PicParm.ColorTable** after the return from REQ\_EXEC.

If **F\_Bmp** is set in **PicParm.Flags**, then the output image rows are padded to a 32-bit (DWORD) boundary and the output image is bottom up. Otherwise, set **F\_Raw** in **PicParm.Flags** and set **RF\_TopDown** in **u.J2K.Region.Flags** if desired.

If **F\_Raw** is set, set **Region.Stride** if desired after RES\_DONE is returned from REQ\_INIT and before calling REQ\_EXEC. If set, be sure to adjust **StripSize** to correspond and make sure the **Put** queue size is at least as big as the adjusted **StripSize** as:

```
StripSize = ( StripSize / Region.Stride ) * <NewStride>;
Region.Stride = <NewStride>;
```

Set **RF\_MakeGray** in **u.J2K.Region.Flags** to convert the color map to the equivalent gray colors and apply that gray color map to the color indices to result in a uniform grayscale output image.

Set **RF\_CM2RGB** in **u.J2K.Region.Flags** to apply the color map to the color indices to result in a 24-bit RGB output image or 8-bit uniform gray output image depending on whether all color map colors are gray. If some color map color is not gray, set **RF\_SwapRB** in **u.J2K.Region.Flags** before REQ\_EXEC to decompress to interleaved RGB samples ordered red, green, and then blue. Otherwise the output interleaved RGB samples are ordered blue, green, and then red. Set **PF2\_KeepColorTable** in **u.J2K.PicFlags2** to optionally have **OP\_J2KE** return the color map in **PicParm.ColorTable** that it would otherwise not return when **RF\_CM2RGB** is set. Any returned color map colors are ordered according to **RF\_SwapRB**.

## Gray Images

If the above conditions aren't met for decompressing to an RGB or color-mapped image, and the component bits per sample is no more than 8 bits per sample, then it is decompressed to a grayscale component. Packed pixel output (more than one pixel per byte) is not supported.

If **F\_Bmp** is set in **PicParm.Flags**, then the output image rows are padded to a 32-bit (DWORD) boundary, the output image is bottom up and the output image samples are unsigned, 8-bits-per-sample and one-byte-per-sample. Otherwise, set **F\_Raw** in **PicParm.Flags** and set **RF\_TopDown**, **RF\_2Byte** (and optionally **RF\_BigEndian**) and **RF\_Signed** in **u.J2K.Region.Flags** if desired.

If **F\_Raw** is set, set **Region.Stride** if desired after RES\_DONE is returned from REQ\_INIT and before calling REQ\_EXEC. If set, be sure to adjust **StripSize** to correspond and make sure the **Put** queue size is at least as big as the adjusted **StripSize** as:

```
StripSize = ( StripSize / Region.Stride ) * <NewStride>;
Region.Stride = <NewStride>;
```

## High-Gray Images (9-16 bits per sample)

If the component samples are larger than 8 bits per sample, then the component is decompressed to high-gray samples.

Set **F\_Raw** in **PicParm.Flags** and set **RF\_TopDown**, **RF\_BigEndian**, and **RF\_Signed** in **u.J2K.Region.Flags** if desired.

If `F_Raw` is set, set **Region.Stride** if desired after `RES_DONE` is returned from `REQ_INIT` and before calling `REQ_EXEC`. If set, be sure to adjust **StripSize** to correspond and make sure the **Put** queue size is at least as big as the adjusted **StripSize** as:

```
StripSize = ( StripSize / Region.Stride ) * <NewStride>;
Region.Stride = <NewStride>;
```

## Images with Planar Components

The **u.J2K.Region** fields describe the first component, or first three components if RGB, of an image. For more complicated images, the application can allocate an array of `PARTITION` data structures to be filled in by `OP_J2KE`. Each element of the array describes a single component. The first `PARTITION`, partition number 0, is specified by **u.J2K.Region**, **u.J2K.StripSize** and **PicParm.Put**. Partitions number 1 through **u.J2K.NumOtherPartitions** - 1, each describe a single other component after the component(s) described as partition 0. Partition number 1 is described by **u.J2K.OtherPartitions[0]**, etc.

In the general case, the application won't know how many partitions to allocate for an input image before `REQ_INIT`. After `RES_DONE` is returned from `REQ_INIT`, `OP_J2KE` will return the total number of partitions in **u.J2K.NumPartitions**. Then, if desired, the application can call `REQ_TERM` to prematurely terminate `OP_J2KE` so the correct number of **OtherPartitions** can be allocated by the application. Now the application can start `OP_J2KE` over again with `REQ_INIT` and with the correct number of **OtherPartitions**. If there are fewer **OtherPartitions** than needed by the image, then the extra image components aren't decompressed.

When the **Put** queue is full and more output data for partition 0 is ready, a `RES_PUT_NEED_SPACE` response is returned, **u.J2K.PartitionNum** will be 0, and the new data will be returned in the **PicParm.Put** queue after it is emptied by the application. When a partition N queue is full and more output data is ready for that partition, a `RES_PUT_NEED_SPACE` response is returned, **u.J2K.PartitionNum** will be N, and the new data will be returned in the **u.J2K.OtherPartitions[N].Queue** queue after it is emptied by the application. Each **u.J2K.OtherPartitions[N].Queue** must be at least **u.J2K.OtherPartitions[N].StripSize** bytes in size.

## Images with Subsampled Components

If an image component is subsampled by horizontally or vertically by 8:1 or less, then the subsampling factor is returned in **u.J2K.OtherPartitions[N].Region.Interlace**. Otherwise, the subsampling factors can be determined using the combination of **u.J2K.ImageXOff**, **u.J2K.ImageYOff**, **u.J2K.OtherPartitions[N].Region.Width** and **u.J2K.OtherPartitions[N].Region.Height**. If `PF2_SkipUpsampling` is not set, `OP_J2KE` will return an upscaled version of the component by linearly interpolating intermediate samples. Otherwise, if `PF2_SkipUpsampling` is set, `OP_J2KE` will return the subsampled version of the component directly.

## Decompressing to a Thumbnail Size

The image can be decompressed to a thumbnail size by setting **u.J2K.Thumbnail** to a value other than 0. The decompressed image height and width will be divided by  $2^{u.J2K.Thumbnail}$  and processing time and memory requirements are reduced by very roughly the same order. The returned **Region.Width** and **Region.Height** values will reflect the Thumbnail dimensions.

The returned **ImageWidth** and **ImageHeight** will reflect the dimensions of the full-size image.

## Decompressing to Lower Resolution

If the image was compressed with more than one bitslice layer, then the image can be decompressed to a lower resolution version by setting **u.J2K.Resolution** to a value other than 0. **u.J2K.Resolution** then specifies the number of highest-resolution bitslice layers to be ignored during decompression.

## Decompressing an Image Subrectangle (Cropping)

A subrectangle of the input image can be compressed by setting **F\_Crop** in **PicParm.Flags** and setting **PicParm.CropXoff**, **PicParm.CropYoff**, **PicParm.CropWidth**, and **PicParm.CropHeight** to describe the subrectangle. **PicParm.CropXoff** and **PicParm.CropYoff** are offsets within the image and not within the canvas. Processing time and memory requirements are reduced by very roughly the same proportion as the area of the subrectangle compared to the area of the image.

## Progressive Decompression

For Progressive decompression, the **Q\_EOF** flag is used to specify that the compressed image data in the current **Get** queue is all the image data currently available. In this case, **REQ\_EXEC** returns **RES\_DONE** and the application uses **REQ\_TERM** to end the decompression process. Then as more data is available in the **Get** queue, the **REQ\_INIT/REQ\_EXEC/REQ\_TERM** process is rerun.

## Reduced-Memory Mode

The opcode provides a means to drastically reduce the amount of memory required for decoding, thereby making it possible to decode extremely large images and/or operate where memory is otherwise limited. To use this mode, set the **PF2\_ReducedMemoryMode** flag in **u.J2K.PicFlags2** before **REQ\_INIT**. The application then has the additional choice of whether to support seeking in the input stream or not by appropriately setting the **PF2\_UseResSeek** flag or not in **u.J2K.PicFlags2**. Leaving the flag clear will typically use less memory and therefore is the recommended mode of operation for most images. For best performance, the image being decoded should contain only one layer, the PCRL progression order, and small precincts (128x128 recommended).

If the image had been compressed with more than one tile, then the reduced-memory mode requires that the opcode be able to seek in the input stream. The application must signal its ability to seek by setting the **PF2\_UseResSeek** flag in **u.J2K.PicFlags2** at **REQ\_INIT** and/or **REQ\_EXEC**. If it does not set the flag before **REQ\_INIT**, the application can check for the existence of tiles by looking at the image and tile dimensions and offsets returned by **REQ\_INIT** and then set the **PF2\_UseResSeek** flag before the call to **REQ\_EXEC**. Note that if the application sets this flag at **REQ\_INIT**, then it must remain set at **REQ\_EXEC**.

If seeking is enabled, be sure to use the values in the **PicParm.SeekOffsetLow** and **PicParm.SeekOffsetHigh** fields when responding to a **RES\_SEEK** request from the opcode in order to get the correct seek offset for extremely large images. If seeking is enabled in reduced-memory mode, multithreading within the opcode is not possible and the opcode operates in single-threaded mode irrespective of the value in **PicParm.NumberOfThreadsAllowed**.

### 3D Slice Encoding of Volumetric Data

If the JPEG2000 image is recognized as a 3D slice encoding of volumetric data, the opcode sets `PF2_3D_Slices` in **u.J2K.PicFlags2** and expands the image as one slice per partition. Each partition carries data for one slice and can contain interleaved RGB samples (8 bits per sample; 24 bpp) or grayscale samples (up to 16 bits). Color-mapped images are not allowed. Subsampling is not allowed.

Set **u.J2K.SliceOff** before `REQ_INIT` to the offset of the first slice to expand; the first slice in the image (offset 0) is the default. The number of slices returned is the lesser of the number of partitions provided to the opcode at `REQ_INIT` or the number of slices in the image. The actual number of slices in the encoded image is returned by `REQ_INIT` in **u.J2K.NumPartitions** for the application to use in later invocations as needed.

The input JPEG2000 compressed image can be in the JPEG2000 raw codestream format or in the JPEG2000 JPX file format.

### Metadata

The opcode finds the most appropriate Color Specification Box for the image and if that box contains ICC Profile data, the data will be returned in **u.J2K.ColorSpecICCPProfileData** and its length in **u.J2K.ColorSpecICCPProfileLen** after `REQ_INIT`. Otherwise, the opcode returns the Part-1 enumerated colorspace if it finds one in the Color Specification Box. The memory pointed to by **u.J2K.ColorSpecICCPProfileData** will be freed by the opcode during `REQ_EXEC`. The ICC Profile data will NOT be used when decoding the image.

If the JPEG2000 image contains Resolution Boxes, either Capture Resolution or Display Resolution, and you supply a **J2K\_EXTRA** structure at **PIC\_PARM.opcodeExtraPtr**, then the Horizontal and Vertical resolution values will be returned in **opcodeExtraPtr->CaptureResVert**, **opcodeExtraPtr->CaptureResHorz**, or **opcodeExtraPtr->DisplayResVert**, **opcodeExtraPtr->DisplayResHorz**.

If the JPEG2000 file contains UUID Boxes, UUID Info Boxes or XML Boxes then they will be returned on the **PIC2List** after `REQ_INIT`.

If the opcode knows how to parse a specific UUID Box then the UUID Box will not be on the **PIC2List** but the individual items will be. For instance, the opcode understands IPTC data and if the JPEG2000 file contains a UUID Box with UUID `33c7a4d2-b81d-4723-a0ba-f1a3e097ad38` then the UUID Box will be parsed and IPTC Packets will be returned on the **PIC2List**.

UUID Info Boxes will be broken into **P2PktUUIDDataEntryURL** packets for each UUID in the UUID Info Box.

XML Boxes can be UTF-8 or UTF-16. They will always be NULL terminated. If it is UTF-16 it will start with a Byte Order Marker (BOM) and be doubly NULL terminated.

---

## OP\_LIE3PLUS: Expand Lossless JPEG and IMStar to DIB or RAW

See the [LOSSLESS3:OP\\_LIE3](#) section for additional information. Note that **OP\_LIE3** requires that **ParmVerMinor** be 2.

**OP\_LIE3** is used for expanding images compressed using IMStar as well as for expanding standard lossless JPEG images. **CompMethod** is returned by **OP\_LIE3** according to the compressed image type. **METHOD\_JPEG** is returned for lossless JPEG, **METHOD\_PPMD** for IMStar discrete compression and **METHOD\_LOCO** for IMStar continuous tone compression. **CompOrder** is returned by **OP\_LIE3** if **CompMethod** is **METHOD\_PPMD**, according to the **CompOrder** set when the image was compressed.

For IMStar images, the bit depth of the output DIB is the same as that of the input compressed image – which is the same as the bit depth of the original source DIB which was compressed. For lossless JPEG images, the bit depth of the output DIB is 8 bits per pixel for gray scale images and 24 bits per pixel for RGB images.

**StripSize** will have been set by **OP\_LIE3** after **REQ\_INIT** is complete. The **Put** queue must be at least **StripSize** bytes in length.

If **AllowedBitErr** were set when the image was compressed, then the value used for compression is returned to the application by **OP\_LIE3** for informational purposes only.

See **UniversalCT** and **NumUC** in [LOSSLESS3:OP\\_LIE3](#) for a description of how lossless JPEG pixel gray levels or color intensities can be mapped to new values. This might be used to apply gamma correction to a gray scale image. If no **UniversalCT** is provided, and if the input precision is greater than the output DIB bits per pixel, then the input gray or color intensity levels are scaled to uniform output intensity levels as though a 256-element **UniversalCT** were provided and where the value of each element were the same as the 0-based index of the element. For example, if a 12-bit lossless JPEG gray scale image is expanded, the scaling function described in **LOSSLESS3:OP\_LIE3** is used to scale the 12-bit JPEG input pixel values to the 8-bit DIB output pixel values.

If **PF\_YieldPut** is set in **u.LL3.PicFlags** then **OP\_LIE3** will return a **RES\_PUT\_DATA\_YIELD** response periodically during compression. This would allow an application to progressively display the image being expanded, or to update expansion progress reporting.

**OP\_LIE3** optionally returns comments, application data, watermark or scripts using **PIC2List** packets. Therefore **OP\_LIE3** requires that **ParmVerMinor** equals 2. See the **PIC2List** section of **Accessing Comments and Other Auxiliary Data** in the **PICTools** and **AIMTools Programmer's Guide** for more information about how opcodes return **PIC2List** packets.

If **PF\_HaveWatermark** is set in **u.LL3.PicFlags**, then a watermark would be combined with the image in such a way that the watermark could be recovered when the image is expanded. The watermark is an ASCII string.

Certain other **u.LL3** fields may be useful to an application, but are not required or used by **OP\_LIP3** or **OP\_LIE3**. These fields may be set by the application and are stored with the image by **OP\_LIP3**. The values are retrieved from the image and returned to the application by **OP\_LIE3**. For more information about these fields, see the **LOSSLESS3:OP\_LIE3** section. These fields are: **NumOfPages**, **PageNum**, **DispMethod**, **UserDelay**, **Xoff**, **Yoff**, **PF\_ApplyTransparency** and **Transparent**.

Image parameters are returned in the **Region** structure after **RES\_DONE** is returned from **REQ\_INIT**. Some of these parameters can then be adjusted by your application before calling **REQ\_EXEC** to change the default output formatting of the decompressed image. If **Stride** would exceed 32767, use the **Region2** structure instead of the **Region** structure. See

the [LOSSLESS3:OP\\_LIE3](#) section and the [REGION2](#) section for more information.

To change the default output image **Stride**, set a new **Stride** and adjust **StripSize** as:

```
pp->u.LL3.StripSize =
    NewStride*(pp->u.LL3.StripSize/pp->u.LL3.Region.Stride);
pp->u.LL3.Region.Stride = NewStride;
```

To output a gray image that is not 9 to 16 bits per pixel using two bytes per pixel, set RF\_2Byte in **Region.Flags** and adjust the **Stride** and **StripSize** to correspond.

To set big-endian ordering of output pixel bytes for a two-byte-per-pixel image, either 9 to 16 bits per pixel or RF\_2Byte, set **Region.PixType** to PT\_GRAYM.

By default, an image compressed with a non-zero **AllowedBitErr** will be decompressed to the same bits-per-pixel as the original image but with the discarded bits set to 0. Instead, the image can be decompressed using fewer bits per pixel by decreasing **AllowedBitErr**. If **Region.Bpp** plus **AllowedBitErr** was originally 9 to 16 and becomes 8 or less after decreasing **AllowedBitErr**, then **Stride** and **StripSize** will also need to be adjusted.

Note: This opcode allows decompressing an image previously compressed with PICTools and password protected. See details in the section describing [the PIC\\_PARM KeyField field](#).

## Bad Suffix Images

For 16 bits-per-pixel images, some non-Pegasus lossless JPEG compressors can create non-compliant lossless JPEG images due to a bug caused by their incorrect reading of the lossless JPEG specification. A compliant JPEG decompressor cannot decompress these images, but these non-compliant decompressors exactly reverse the non-compliant compression so they can view the images as expected. Their decompressor bug also means that these decompressors cannot decompress all compliant lossless JPEG 16 bpp images such as are created by OP\_LIP3.

There are so many of these non-compliant images in the field that Pegasus has implemented a decompressor work-around to allow these non-compliant images to be decompressed. Set PF\_IgnoreBadSuffix in u.LL3.PicFlags to decompress an image known to be non-compliant. Be aware, however, that setting this flag for a compliant image may cause the image to be decompressed incorrectly.

It is impossible in principle for OP\_LIE3, or any decompressor, to detect with absolute certainty whether a compressed image was created with a buggy compressor or by a compliant compressor. However, OP\_LIE3 provides a Pegasus-proprietary method that, with high probability, will automatically detect whether the compressed image is compliant or not and then correctly decode it. OP\_LIE3 does this when PF\_AutoIgnoreBadSuffix is set in u.LL3.PicFlags. If non-compliant encoding of the image is detected, OP\_LIE3 will set PF\_IgnoreBadSuffix in u.LL3.PicFlags; if the image was found to be compliant, the flag will be cleared. In the unlikely case that OP\_LIE3 cannot do an automatic detection and return without error or if OP\_LIE3 returns without error but the decoded image appears wrong, do not set PF\_AutoIgnoreBadSuffix

but set or clear `PF_IgnoreBadSuffix` to explicitly force non-compliant or compliant decoding, respectively. If the error persists, then the image is bad.

---

## OP\_S2D: Expand sequential JPEG to DIB or RAW

See the [DIB OUTPUT: OP\\_S2D](#) section for additional information.

### DICOM YBR\_FULL\_422 Output from Color JPEG Image Input

To decompress a color JPEG input image to a YBR\_FULL\_422 decompressed output image:

- In `J2D.PicFlags2` set `PF2_Cosited`.
- In `J2D.PicFlags2` set `PF2_SmoothUpsample`.
- Set `J2D.YuvOutputType` to `BI_YUY2` or `BI_UYVY`
- Set `DibSize` to `16`.

### DICOM YBR\_PARTIAL\_422 Output From Color JPEG Image Input

To decompress a color JPEG input image to a YBR\_FULL\_422 decompressed output image:

- In `J2D.PicFlags2` set `PF2_Cosited`.
- In `J2D.PicFlags2` set `PF2_SmoothUpsample`.
- Set `J2D.YuvOutputType` to `BI_YUY2` or `BI_UYVY`
- Set `DibSize` to `16`.
- In `J2D.PicFlags2` set `PF2_CompressedYuv`

### Color JPEG input image

#### A. Creating a 16, 24 or 32 bit color DIB

- `u.J2D.DibSize` is set to 16, 24 or 32.
- `PF_ConvertGray` is not set.
- `PF_Dither` is not used for a 24 or 32-bit DIB. It may be set for a 16-bit DIB if desired.
- `PF_CreateDibWithMadeColors` is not used.
- `PF_UseOnlyMadeColors` is not used.
- `PF_MakeColors` may be used to create an optimum color table (see case B. following), otherwise `u.J2D.PrimClrToMake`, `u.J2D.SecClrToMake` and `u.J2D.MadeColorTable` are not used and `RES_COLORS_MADE` will not be returned.
- `u.J2D.GraysToMake` is not used.
- If R-G-B output is desired, Set `PF_SwapRB`

#### B. Creating a 4 or 8 bit color DIB with optimum colors

- `u.J2D.DibSize` is set to 4 or 8.
- `PF_ConvertGray` is not set.
- `PF_Dither` may be set as desired.

- **PF\_MakeColors** is set.
- **PF\_CreateDibWithMadeColors** is set.
- **PF\_UseOnlyMadeColors** is set unless you want to display an intermediate image using the color table saved with the JPEG image, if any, (see case C. following) or using another color table supplied in **ColorTable** (see case D. following).
- **PrimClrToMake** and **SecClrToMake** may be specified. Typically, **PrimClrToMake** will be set to 16 or 256 and **SecClrToMake** will be 0.
- **RES\_COLORS\_MADE** will be returned by **Pegasus** after the optimum colors are created.
- A buffer sufficiently large for the created colors can be allocated and **MadeColorTable** set to the buffer address, if desired. If so, then the optimum colors are copied to **MadeColorTable** in addition to being copied to **ColorTable**.
- **Head.biClrUsed**, **Head.biClrImportant** and **ColorTable** are ignored as input and are set to the created palette, primary or secondary, which is appropriate to **u.J2D.DibSize** on output.

### C. Creating a 4 or 8 bit color DIB with an external palette

- **u.J2D.DibSize** is set to 4 or 8.
- **PF\_ConvertGray** is not set.
- **PF\_Dither** may be set as desired.
- **PF\_MakeColors** is not set.
- **PF\_CreateDibWithMadeColors** is not set.
- **PF\_UseOnlyMadeColors** is not set.
- **PrimClrToMake** and **SecClrToMake** are not set.
- **RES\_COLORS\_MADE** will not be returned.
- **Head.biClrUsed**, **Head.biClrImportant** and **ColorTable** are set on input according to the external palette.
- **MadeColorTable** is not used.
- Pegasus sets **Head.biClrUsed** to the number of colors in the primary or secondary palette which is appropriate to **u.J2D.DibSize**. **Head.biClrImportant** is set equal to **Head.biClrUsed** and the color values are copied to **ColorTable**.

### D. Creating a 4 or 8 bit gray scale DIB

- **u.J2D.DibSize** is set to 4 or 8.
- **PF\_ConvertGray** is set.
- **PF\_Dither** may be set as desired.
- **PF\_MakeColors** may be set if **ColorTable** has a palette, but you wish to create a palette. (You could also set **Head.biClrUsed** to 0 to force colors to be made).
- **PF\_CreateDibWithMadeColors** is not set.
- **PF\_UseOnlyMadeColors** is not set.
- **GraysToMake** is set unless you want to use the default according to **u.J2D.DibSize** or unless you are supplying a gray scale color table in **ColorTable**.
- **RES\_COLORS\_MADE** will not be returned.
- **Head.biClrUsed**, **Head.biClrImportant** and **ColorTable** are set if you want to use that gray scale color table instead of creating one. These values are overridden by **PF\_MakeColors**.
- **MadeColorTable** is not used.

- If gray levels were made, **Pegasus** sets **Head.biClrUsed** and **Head.biClrImportant** to the number of gray levels made and sets **ColorTable** according to the gray levels made.

## E. Creating a YUV Output Image

- **u.J2D.DibSize** is set to 16
- **Set PF\_YuvOutput**
- **Set YuvOutputType** to **BI\_YUY2** or **BI\_UYVY**.

## CMYK Color JPEG input image

CMYK JPEG Images may only be decompressed to interleaved CMYK

- **u.J2D.DibSize** is set to 32.
- **PF\_ConvertGray** is not set.
- **PF\_Dither** is not used.
- **PF\_CreateDibWithMadeColors** is not used.
- **PF\_UseOnlyMadeColors** is not used.
- **PF\_MakeColors** is not used to be returned.
- **.u.J2D.GraysToMake** is not used.
- **Set PF2\_NoYccTransform**.

## Gray scale JPEG input image

### A. Input is 12 bit gray scale JPG

- **PegasusQuery** will return **Head.biBitCount** = 12.
- **Set J2D.PicFlags** |= **PF\_NoDibPad** to avoid any padding of output lines otherwise output lines are padded to a multiple of 4 bytes
- To set the width of the output line, set **J2D.PicFlags** |= **PF\_WidthPadKnown**, and set **J2D.WidthPad** to the desired width in bytes.

#### 1. Create a 12 bit image

- set **J2D.DibSize** = 16
- If **J2D.GrayMap12Bit** is Null, then the 12 bits for each pixel of the decompress image will be placed in the low-order 12 bits of the 16 bit field.
- If **J2D.GrayMap12Bit** is **NOT** Null, then it is assumed to be a table of 4096 16-bit words

#### 2. Create an 8 bit image

- set **J2D.DibSize** = 8
- **J2D.GrayMap12Bit** **MUST NOT** be Null, it is assumed to be a table of 2<sup>12</sup> 8-bit bytes.
- **OR** Set **S2D.PicFlags** |= **PF\_Quickview12**, output will be the top 8 bits of the 12-bit field; it is less accurate because the low bits are thrown away during dequantization, but it is very fast.

## Notes – Using J2D.GrayMap12Bit

The lookup table **J2D.GrayMap12Bit** may be constructed to map the 12 bits to either a 16-bit or 8-bit field in any way desired. The function **BuildTranslationTable12** which is included in the test program **jetest.c** illustrates this for a choice of brightness, contrast, gamma-correction, output bits of precision, and effective input bits of precision. Ordinarily the effective input bits

of precision would be 12, since the data was supposedly 12-bit. But the user may know, for example, that the original data was really 10 bits right-justified. The jpeg decompressor does not know this, and since it is lossy, some of the decompressed data may exceed 10 bits. So the user should set  $\text{Num} = 2^{10}$  in the input of `BuildTranslationTable12`. The table will then accept inputs up to 12 bits, but the input values  $\geq 2^{10}$  will be clipped to have the same output as  $2^{10} - 1$  has. This is the option `-VS#` in the command line test program; in this example `#` would be set to 10. For example if `-VP8` and `-VS10` are set in the command-line test program, the high 8 bits of the alleged 10-bit data are mapped to the output bytes, and anything above 10 bits gets mapped to the same value as  $2^{10} - 1$  does.

## B. Input is 8 bit gray scale JPG - Creating a 4 or 8 bit gray scale DIB

- **u.J2D.DibSize** is set to 4 or 8.
- **PF\_ConvertGray** may be set.
- **PF\_Dither** may be set as desired.
- **PF\_MakeColors** may be set if **ColorTable** has a palette, but you wish to create a palette. (You could also set **Head.biClrUsed** to 0 to force colors to be made).
- **PF\_CreateDibWithMadeColors** is not set.
- **PF\_UseOnlyMadeColors** is not set.
- **PrimClrToMake** and **SecClrToMake** are not set.
- **GraysToMake** is set unless you want to use the default according to **u.J2D.DibSize** or unless you are supplying a gray scale color table in **ColorTable**.
- **RES\_COLORS\_MADE** will not be returned.
- **Head.biClrUsed**, **Head.biClrImportant** and **ColorTable** are set if you want to use that gray scale color table instead of creating one. These values are overridden by **PF\_MakeColors**.
- **MadeColorTable** is not used.
- If gray levels were made, **Pegasus** sets **Head.biClrUsed** and **Head.biClrImportant** to the number of gray levels made and sets **ColorTable** according to the gray levels made.
- If the JPEG image were saved with a color table, and the color table has 256 colors, then the green components of the colors are used as for gamma correction of the created or supplied gray levels. There is no way to override this behavior.

## Notes

- The **Put** queue is not accessed during `REQ_INIT`.
- The **Put** queue can be allocated during the `REQ_EXEC` response loop if desired.
- The **Put** queue size must be at least **u.J2D.StripSize** and must be an integer multiple of `StripSize` when using `Q_REVERSE`.
- The **Get** queue size must be at least 128 bytes.
- The **Get** queue must be allocated prior to `REQ_INIT`, although it need not contain any data until requested by `RES_GET_NEED_DATA`.
- `REQ_INIT` changes the `PIC_PARM` data in such a way that an `OP_S2D` operation cannot be restarted after an error by simply correcting the error and using the same `PIC_PARM` data.
- This opcode allows decompressing an image previously compressed with AIMTools and password protected. See details in the section describing [the PIC\\_PARM KeyField field](#).

## Alternative Legacy Approach for Retrieving Comments and Application Data: Separate Comment and AppField Buffers.

OP\_S2D and OP\_P2D support an alternative approach for retrieving comments and other data from an image and making it available to the application. This is a legacy approach and is more restrictive than the use of the newer PIC2List functions. When using this approach, an application would allocate a buffer where the data of interest will be eventually placed by Pegasus. There are two possibilities according to when the buffer space is allocated.

### Application Pre-allocates a Buffer before Pegasus Operation

Note: This technique is also supported by PegasusQuery. In fact, it is the only choice for retrieving comments and other data via **PegasusQuery**.

In most cases, the application doesn't know how much data may be included in the comment or application data. In some cases an application either knows a probable maximum size or knows that it is acceptable to truncate the data over some maximum size. In these cases, the application may choose to pre-allocate a buffer for the data before calling **Pegasus** for initialization.

The application sets PIC\_PARM.Comment (PIC\_PARM.u.S2D.AppField) to the address of the buffer. PIC\_PARM.CommentSize (PIC\_PARM.u.S2D.AppFieldSize) is set to the allocated size of the buffer. When **Pegasus** encounters comment data or application data in the image, it copies as much of the data to the appropriate buffer as will fit. PIC\_PARM.CommentLen (PIC\_PARM.u.S2D.AppFieldLen) is set to the length of the copied data.

The application data should appear only once in the image. Comment data may appear more than once. When the application pre-allocates a comment buffer according to this section, the last comment data encountered prior to **Pegasus** returning a response is the comment data which appears in the PIC\_PARM.Comment buffer any other prior comments are lost.

### Application Allocates a Buffer during Pegasus Operation

When an application does not know the size of the image comment data, and must retrieve all available comment data, the application sets PIC\_PARM.CommentSize to 0 and sets the PF\_AlllocateComment flag in the PicFlags field of the operation-specific PIC\_PARM union structure.

For application data, the application sets the PIC\_PARM.u.S2D.AppFieldSize field to -1 and doesn't set a PicFlags flag.

When **Pegasus** encounters the comment (application data) it sets the PIC\_PARM.CommentLen (PIC\_PARM.u.S2D.AppFieldLen) field and returns to the application with a response of RES\_ALLOCATE\_COMMENT\_BUF (RES\_ALLOCATE\_APP2\_BUF). At this time the application can allocate a buffer and set PIC\_PARM.Comment (PIC\_PARM.u.S2D.AppField) and PIC\_PARM.CommentSize (PIC\_PARM.u.S2D.AppFieldSize). The comment is retrieved when the operation is continued by returning 0 from the application's DeferFn function.

**Pegasus** returns the RES\_HAVE\_COMMENT response when the PF\_AllocateComment flag is set, and an image comment is encountered, and an image comment was previously retrieved. This response gives the application an opportunity to save the previous comment before overwriting it by retrieving the current comment. When the application continues the operation after saving the previous comment, a RES\_ALLOCATE\_COMMENT\_BUF response is returned so a buffer can be allocated for the current comment.

The following example demonstrates this by storing the first 100 comments and their length in two, previously allocated, arrays: CommentsData and CommentsLen.

```

PicParm.u.S2D.PicFlags |= PF_AllocateComment;
PicParm.u.S2D.AppFieldSize = -1;
resp = Pegasus(&PicParm, REQ_INIT);
if ( resp == RES_DONE )
    resp = Pegasus(&PicParm, REQ_EXEC);
if ( resp == RES_DONE )
    Pegasus(&PicParm, REQ_TERM);
////////////////////////////////////
LONG DeferFn(PIC_PARM* pp, RESPONSE resp)
{
    switch (resp) {
        case RES_GET_NEED_DATA:
            Status = GetData(pp);
            break;
        case RES_PUT_NEED_SPACE:
            Status = PutSpace(pp);
            break;
        case RES_ERR:
            Status = pp->Status;
            break;
        case RES_ALLOCATE_APP2_BUF:
            pp->u.S2D.AppFieldSize = pp->u.S2D.AppFieldLen;
            pp->u.S2D.AppField = malloc(pp->u.S2D.AppFieldSize);
            if ( pp->u.S2D.AppField == 0 ) {
                pp->Status = ERR_OUT_OF_SPACE;
                return ( 1 ); // abort the operation
            }
            break;
        case RES_HAVE_COMMENT:
            if (ncomments < 100) {
                CommentsData[ncomments] = pp->Comment;
                CommentsLen[ncomments++] = pp->CommentLen;
            }
            pp->CommentSize = 0;
            break;
        case RES_ALLOCATE_COMMENT_BUF:
            pp->CommentSize = pp->CommentLen;
            pp->Comment = malloc(pp->CommentSize);
            if ( pp->Comment == 0 ) {
                pp->Status = ERR_OUT_OF_SPACE;
                return ( 1 ); // abort the operation
            }
            break;
    }
    // abort if Status != ERR_NONE
    return ( Status != ERR_NONE );
}

```

---

## OP\_ZOOM2: DIB zoom/shrink, gray scale, halftone

The OP\_ZOOM2 opcode operates on input DIB's creating output DIB's of various width, height, bit-depth and pixel type.

Augural zooming technology provides enhanced edge sharpness.

Allowable input/output pixel types are:

- Grayscale from 1 to 16 bits, packed to 1, 8, or 16 bits per pixel in big-endian or little-endian format;
- Color-mapped from 1 to 8 bits, packed to 1, 4, or 8 bits per pixel;
- RGB at 15 (packed to 16 bits per pixel), 16, or 24 bits.
- RGB/Alpha at 32 bits per pixel.
- YUYV at 16 bits per pixel.

Optional conversion of the image to grayscale is available for all input/output combinations. Conversion to grayscale is mandatory for certain input/output bit-depth/pixel-type combinations.

Optional fast zoom mode uses resampling to provide faster execution, but with poorer image quality. Fast zoom mode is mandatory for certain input/output bit-depth/pixel-type combinations.

See the [ZOOM2\\_PARMS](#) structure section for additional information.

### Notes

- **Get** and **Put** queues are not used during the REQ\_INIT request.
- The **Get** queue can be allocated during the REQ\_EXEC response loop if desired.
- The **Put** queue can be allocated during the REQ\_EXEC response loop if desired.
- **Get** and **Put** buffers must be large enough to hold one complete image row.
- RGB/Alpha output is possible only with RGB/Alpha input.
- The output image stride and offset may be set between the REQ\_INIT request and the REQ\_EXEC request.

---

## PLUS Opcodes – Beyond 8bpp Grayscale Support

Several of the image format opcodes have had support added for extended image data. These opcodes are denoted by a trailing PLUS in the opcode name. Each of these opcodes is a superset of the standard opcode and the addition of support for extended image data. Generally these opcodes are utilized exactly as their standard version.

### A. OP\_LIE3PLUS

The PLUS version of the lossless decompressor will decompress all image data compressed using LIP3PLUS and images compressed as planar with RGB planes. See additional information on the lossless decompressor on the [OP\\_LIE3](#) section.

### B. OP\_SE2DPLUS

The PLUS version of the lossy JPEG decompressor adds the ability to handle grayscale data up to 12bpp inclusive and the ability to decompress cosited color 4-2-2 (= 2-1-1) subsampled JPEG data. See additional information in the [OP\\_S2D](#) section.

## General Structures

---

### BITMAPINFOHEADER

```
typedef struct {
    DWORD  biSize;
    LONG   biWidth;
    LONG   biHeight;
    WORD   biPlanes;
    WORD   biBitCount;
    DWORD  biCompression;
    DWORD  biSizeImage;
    LONG   biXPelsPerMeter;
    LONG   biYPelsPerMeter;
    DWORD  biClrUsed;
    DWORD  biClrImportant;
} BITMAPINFOHEADER;
```

The PIC\_PARM **Head** field is a BITMAPINFOHEADER. This is the identical structure defined in Windows. The AIMTools operations have extended the values available for biCompression and they have slightly overloaded the meaning of biClrImportant as it is related to biClrUsed. AIMTools operations also fully initialize the BITMAPINFOHEADER fields. Windows allows some fields to be 0 if the correct value can then be computed using other fields. Therefore, a BITMAPINFOHEADER in a file not created by AIMTools operations might have values which are not fully consistent with the following descriptions.

#### **biSize**

Set to sizeof(BITMAPINFOHEADER).

#### **biWidth**

Width of the image in pixels.

#### **biHeight**

Height of the image in pixel lines.

#### **biPlanes**

Always 1.

#### **biBitCount**

The number of bits per pixel. Standard Windows .BMP files allow values of 1, 4, 8, 16, 24 and 32. 16-bit Windows has difficulty with 16 and 32. Not all values are permitted by all AIMTools operations. The number of bits per pixel is directly related to the maximum possible number of colors per pixel. As each bit can represent 2 colors, the maximum number of colors is 2 raised to the **biBitCount** power—except the 32-bits/pixel format reserves the high-order 8 bits for other purposes which are not standardized. Thus the number of colors for 32 bits per pixel is ordinarily regarded as the same as the number of colors for 24 bits per pixel.

**biCompression**

This value indicates the format of the image. The possible values as extended by the AIMTools operations are:

<b>Value</b>	<b>Meaning</b>
BI_BMPO	Old-style BMP uncompressed (and OS/2 1.x)
BI_BMPR	New-style BMP RLE
BI_CALS	CALS image
BI_C4	C4 image
BI_CMYK	CMYK 4-byte interlaced
BI_DCX	Generic DCX
BI_DCXZ	DCX multi-image
BI_EMF	Enhanced Windows Metafile
BI_G3	Raw G3 Fax 1D
BI_G32D	Raw G3 Fax 2D
BI_GIFI	GIF interlaced
BI_GIFN	GIF non-interlaced
BI_GIFu	GIF unknown
BI_GR12	Gray 12 bpp, uncompressed, low-order bits in 16 bit field
BI_HDP	JPEG XR/ HDPhoto File
BI_IYUV	MS Video IYUV subtype
BI_I420	MS Video I420 subtype
BI_ICON	.ICO image
BI_IOCA	IOCA image
BI_J2K	JPEG2000 image
BI_JB2	JBIG2 image
BI_JLS	JPEG-LS
BI_JPEG	Sequential JPEG compressed
BI_JPGE	Sequential JPEG PIC-enhanced compression
BI_JPGL	Lossless JPEG
BI_MDCA	MO:DCA™
BI_OS2	OS2 2.x+ uncompressed
BI_PCD	PhotoCD generic
BI_PCX	PCX generic
BI_PCX1	PCX 1-bit
BI_PCX2	PCX 2-bit (CGA) (color map)
BI_PCX3	PCX 3-bit (EGA) (color map)
BI_PCX4	PCX 4-bit (color map)
BI_PCX8	PCX 8-bit (color map)
BI_PCXT	PCX 24-bit
BI_PBM	Portable Bitmap 1bpp
BI_PBMA	ASCII Portable Bit Map up to 1bpp
BI_PDF	PDF wrapper around images
PI_PGM	Portable Bit Map up to 8bpp Gray
PI_PGMA	ASCII Portable Bit Map up to 8bpp Gray
BI_PIC1	PIC-format IM1 compressed
BI_PIC4	PIC-format IM4 compressed
BI_PIC8	PIC-format IM8 compressed
BI_PICG	G3/G4
BI_PICJ	PIC-format sequential JPEG
BI_PICL	PIC-format IMStar compressed
BI_PICP	PIC-format progressive JPEG

BI_PJPG	Progressive JPEG compressed
BI_PNG1	PNG monochrome non-interlaced
BI_PNG2	PNG gray scale non-interlaced
BI_PNG3	PNG color map non-interlaced
BI_PNG4	PNG rgb non-interlaced
BI_PNG5	PNG monochrome interlaced
BI_PNG6	PNG gray scale interlaced
BI_PNG7	PNG color map interlaced
BI_PNG8	PNG rgb interlaced
BI_PNM	Portable Any Map (generic name)
BI_PPM	Portable Bit Map 24bpp RGB
BI_RGB	Windows .BMP uncompressed
BI_RGBA	RGBA ordered B/G/R/A
BI_RGB565	RGB 5-6-5
BI_TGA	Targa generic
BI_TGA1	Targa type 1 (color map) uncompressed
BI_TGA2	Targa type 2 (bgr) uncompressed
BI_TGA3	Targa type 3 (gray) uncompressed
BI_TGA9	Targa type 9 (color map) RLE
BI_TGAA	Targa type 10 (bgr) RLE
BI_TGAB	Targa type 11 (gray) RLE
BI_TIF	TIFF generic
BI_TIF1	TIFF 1-bit
BI_TIFC	TIFF 24-bit (rgb)
BI_TIFG	TIFF up to 8-bit (gray scale)
BI_TIFJ	TIFF JPEG
BI_TIFK	TIFF CMYK
BI_TIFL	TIFF LZW
BI_TIFM	TIFF up to 8-bit (color map)
BI_TIFu	TIFF unknown
BI_TIFZ	TIFF multi-image
BI_UYVY	MS Video UYVY subtype
BI_YUY2	MS Video YUY2 subtype
BI_YV12	MS Video YV12 subtype
BI_WAVE	Wavelet
BI_WAVP	Progressive Wavelet
BI_WBMP	Wireless Bitmap
BI_WMF	Windows Metafile
BI_WSQ	WSQ compressed

**biSizeImage**

This value is the size in bytes of the image data. Each pixel line may be padded out to some boundary, frequently a DWORD boundary, so computing the **biSizeImage**, even for a DIB, is not necessarily as straightforward as multiplying the width times the height times the bits per pixel and dividing by 8 bits per byte.

**biXPelsPerMeter****biYPelsPerMeter**

These values are stored and retrieved by the AIMTools operations.

**biClrUsed**

This is the number of RGBQUAD elements in the **ColorTable** associated with the image. If there is no **ColorTable** associated with the image, this will be 0.

### **biClrImportant**

This is the number of RGBQUAD elements in the **ColorTable** which are used in the image. The first **biClrImportant** elements of **ColorTable** are used. **biClrImportant** is always less than or equal to **biClrUsed**.

Some AIMTools operations overload the meaning slightly. If **biClrImportant** is less than **biClrUsed**, then the colors from the **ColorTable** element whose index is **biClrImportant** and extending to the end of the **ColorTable**, can be used as a secondary **ColorTable**. This allows optimal color tables to be determined based on two different image bit depths and allows both to be embedded in the image.

---

## PIC\_PARM

```

typedef struct {
    PICINTPTRT      Reserved0;
    LONG            ParmSize;
    BYTE            ParmVer;
    BYTE            ParmVerMinor;
    WORD            Reserver1;
    LONG            Status;
    OPERATION       Op;
    BITMAPINFOHEADER Head;
    RGBQUAD         ColorTable[272];
    LONG            PicVer;
    ORIENTATION     VisualOrient;
    LONG            CommentSize;
    LONG            CommentLen;
    CHAR PICHUGE*   Comment;
    LPARAM          App;
    LONG            PercentDone;
    WORK_AREA PICHUGE* Reserved;
    QUEUE           Get;
    QUEUE           Put;
    BYTE            KeyField[8];
    LONG            (*DeferFn)(struct PIC_PARM *, RESPONSE);
    DWORD           Flags;
    DWORD           Flags2;
    WORD            CropWidth;
    WORD            CropHeight;
    WORD            CropXoff;
    WORD            CropYoff;
    DWORD           ImageNumber;
    DWORD           PacketType;
    DWORD           SeekInfo;
    CHAR PICHUGE*   PIC2List
    LONG            PIC2ListSize;
    LONG            PIC2ListLen;
    REGION          RegionIn;
    REGION          RegionOut;
    WORD            OpVersion0;
    WORD            OpVersion1;
    WORD            OpVersion2;
    WORD            OpVersion3;
    BYTE PICHUGE*   stkReserved;
    BYTE PICHUGE*   opcodeExtraPtr;
    BYTE PICHUGE*   ReservedPtr3;
    BYTE PICHUGE*   ReservedPtr4;
    BYTE PICHUGE*   ReservedPtr5;
    BYTE PICHUGE*   ReservedPtr6;
    BYTE PICHUGE*   ReservedPtr7;
    BYTE PICHUGE*   ReservedPtr8;
    CHAR PICHUGE*   LoadPath;
    PICINTPTRT      LoadResInstance;
    struct PIC_PARM_TAG PICHUGE* NestPP;

```

REQUEST	NestReq;
PICINTPTRT	tlsReserved;
PICINTPTRT	tlsReserved2;
PICINTPTRT	memReserved;
PICINTPTRT	wrapReserved;
PICINTPTRT	wrapReserved2;
WORD	DispVersion0;
WORD	DispVersion1;
WORD	DispVersion2;
WORD	DispVersion3;
PICINTPTRT	DispExports;
PICINTPTRT	tlsReserved3;
PICINTPTRT	tlsReserved4;
PICSIZE	IOCropXoff;
PICSIZE	IOCropYoff;
PICSIZE	IOCropWidth;
PICSIZE	IOCropHeight;
PICINTPTRT	tlsReserved5;
PICINTPTRT	tlsReserved6;
PICINTPTRT	NumberOfThreadsAllowed;
PICINTPTRT	NumberOfThreadsUsed;
PICINTPTRT	SeekOffsetLow;
PICINTPTRT	SeekOffsetHigh;
PICINTPTRT	Reserveds[88];

```

union {
    PEGQUERY QRY;
    DIB_INPUT D2J;
    DIB_OUTPUT J2D;
    TRANS2P S2P;
    TRANP2S P2S;
    D2F_STRUC D2F;
    F2D\_STRUC F2D;
    UTL_STRUC UTL;
    LOSSLESS3 LL3;
    CAMERARAWE CRE;
    REORIENT ROR;
    ZOOM\_PARMS ZOOM;
    WAVELET WAVE;
    PNG_UNION PNG;
    WSQ_UNION WSQ;
    TIFF_EDIT TED;
    MODCA_UNION Modca;
    CLEAN CLN;
    SCANFIX SF;
    IDP_UNION IDP;
    J2K\_UNION J2K;
    J2KT_UNION J2KT;
    SB_PARMS SB;
    JLS_UNION JLS;
    PDF_UNION PDF;
    JBIG2_UNION JBIG2;
    JPIP_UNION JPIP;
    ADJUST ADJ;
    CAD_UNION CAD;
    HDP_UNION HDP;

```

```

        DIB_INPUT      D2S;
        DIB\_OUTPUT      S2D;
        DIB_OUTPUT      P2D;
        INTERNAL        Reserved;
    } u;
} PIC_PARM;

```

The **PIC\_PARM** structure has information to control the **Pegasus** and **PegasusQuery** operations. This structure is divided into two parts: information common to all or most operations, and information specific to a particular operation. Depending on the operation, each field of the **PicParm** structure may be: an input parameter describing attributes of the operation to be performed, an output parameter describing the results of the operation, both an input and output parameter, or not used.

We recommend “zeroing” the contents of this structure before setting any fields for the operation.

## Fields

### Reserved0

Reserved. The field should be set to 0 by the application.

### ParmSize

Specifies the size of the **PicParm** structure. The value is used by the operation to verify that this is the proper structure. The field is input by the application.

### ParmVer

Specifies the current major version of the **PicParm** structure. Currently it must be set to CURRENT\_PARMVER. This field is input by the application.

### ParmVerMinor

Specifies the current minor version of the **PicParm** structure. The default is 1. On a per-opcode basis, this may be set to other than 1. This document contains the appropriate value for ParmVerMinor for each opcode in the usage section for each opcode. The changes.txt file will call out any change to this value required by any of the opcodes. This field is input by the application.

### Reserver1

Reserved. The field should be set to 0 by the application.

### Status

Specifies the error value when RES\_ERR is returned. **Status** has the value ERR\_NONE unless RES\_ERR is returned. A negative value is returned for **Status** when an error occurs. See ERRORS.H for descriptive names for the error numbers. Positive values are reserved for use by the application. This field is output by **Pegasus**.

**Op**

Specifies the specific operation to be performed. This field is input by the application. The possible values for this field are:

<b>Opcode</b>	<b>Meaning</b>
OP_ADJUST	Adjust image color and highlights
OP_BINARIZE	Convert color or grayscale image to black and white
OP_CAD2D	Convert CAD to DIB
OP_CAMERARAW	Expand a RAW digital camera file
OP_CLEAN	Correct deficiencies or enhance image features
OP_D2F	DIB transformed to supported image file format
OP_D2FPLUS	OP_D2F plus GIF
OP_D2J	DIB compressed to sequential or Progressive JPEG
OP_D2MDCA	Compress to MODCA / CALS
OP_D2PDF	Inserts a DIB into a PDF file (SJPEG, G3/4, JBIG2)
OP_D2S	DIB compressed to sequential JPEG
OP_D2SE	DIB compressed to enhanced sequential JPEG
OP_D2SEPLUS	OP_D2SE plus compress to 12-bit JPEG
OP_D2W	DIB compressed using wavelet
OP_EXP4	IM4 expanded to DIB
OP_F2D	Supported image file format transformed to DIB
OP_F2DPLUS	OP_F2D plus GIF and TIFF/LZW
OP_HDPHOTOE	Microsoft HD Photo Expand
OP_HDPHOTOP	Microsoft HD Photo Pack
OP_JBIG2E	Expand JBIG2
OP_JBIG2P	Compress to JBIG2
OP_J2KE	JPEG2000 Decompression
OP_J2KE3D	JPEG2000 Decompression with 3D
OP_J2KERGB	JPEG2000 Decompression limited to RGB and 8bpp gray
OP_J2KP	JPEG2000 Compression
OP_J2KP3D	JPEG2000 Compression with 3D
OP_J2KPRGB	JPEG2000 Compression limited to RGB and 8bpp gray
OP_J2KTRANSCODE	JPEG2000 Transcode
OP_JLSE	JPEG LS Decompression
OP_JLSP	JPEG LS Compression
OP_JPIPCLIENT	JPIP client
OP_JPIPSERVER	JPIP server
OP_LIE3	IMStar or lossless JPEG expanded to DIB
OP_LIE3PLUS	IMStar or 16-bit lossless JPEG expanded to DIB
OP_LIP3	DIB compressed to IMStar or lossless JPEG
OP_LIP3PLUS	DIB compressed to IMStar or 16-bit lossless JPEG
OP_MDCA2D	MODCA / CALS Expand
OP_P2D	Progressive JPEG expanded to DIB
OP_P2S	Progressive JPEG transformed to sequential JPEG
OP_PDF2D	Extract a DIB from a PDF file (SJPEG, G3/4, JBIG2)
OP_ROR	Sequential JPEG lossless rotation/inversion/enhancement
OP_PNGE	PNG expanded to DIB
OP_PNGP	Compress to PNG
OP_RIDP2	Receive IDP – supports J2K
OP_RORE	EPIC lossless rotation/inversion/enhancement
OP_S2D	Sequential JPEG expanded to DIB
OP_SE2D	Enhanced Sequential JPEG expanded to DIB
OP_SE2DPLUS	OP_SE2D plus expand 12-bit JPEG
OP_S2P	Sequential JPEG transformed to progressive JPEG

OP_TIDP2	Transmit IDP – supports J2K
OP_UTL	Color reduction utility
OP_W2D	Wavelet expanded to DIB
OP_WSQE	Expand WSQ format
OP_WSQP	Compress to WSQ format
OP_ZOOM	DIB size changed or converted to grayscale/halftone
OP_ZOOM2	DIB size changed or converted to grayscale/halftone

### Head

Specifies a **BITMAPINFOHEADER** structure, as defined by Windows, that contains information about the dimensions and color format of the image. In general, **Head** is input by the application and is also output by **Pegasus**.

When a DIB is the result of an operation, **Head** is output by **Pegasus** for the DIB before the first pixel of the DIB is returned from the operation, typically by the time **REQ\_INIT** is finished. When a DIB is the source for an operation, **Head** is input by the application and must describe the DIB before **Pegasus** is called with a **REQ\_INIT** request. For other operations, it may be input or output or both. See the operation-specific documentation.

**Head.biClrUsed** and **Head.biClrImportant** refer to the following **ColorTable** field. Therefore, and since **ColorTable** immediately follows **Head**, a pointer to this field can be treated as a pointer to a **BITMAPINFO** structure, for example for use as a parameter to a call to the Windows function **SetDIBitsToDevice**.

### ColorTable

Specifies an array of **RGBQUAD** structures that specify up to 256 primary colors and up to 16 secondary colors. **ColorTable** is input or output or both in similar circumstances and for similar operations as the **Head** field. See the operation-specific documentation.

### PicVer

Specifies the version number of the PIC algorithm used to create the PIC image. The PIC algorithms place an algorithm version number into the image when compressing an image. When decompressing an image, this number is retrieved from the image to ensure the proper algorithm is applied. The difference between this field and the **ParmVer** field is that **ParmVer** specifies the **PIC\_PARM** structure version. This field is output by **Pegasus**.

### VisualOrient

This field is provided for use by the application. It is intended to be descriptive of the image orientation when displayed. It is not used by the AIMTools operations except that they allow the value to be stored with the image and retrieved from the image. This field is input by the application for operations creating an image format which supports the field. It is output by **Pegasus** for operations which take as input an image format which support the field.

Note that it is not always obvious when an image is rotated 180° clockwise versus when the image is inverted by being reflected across the image horizontal midpoint. In the first case, you could read text in the image by standing on your head. In the second case, you could read text in the image by standing on your head and looking at the image in a mirror. (Please don't try this at home.) Combining the two yields an image which is right-side up, but mirror-reversed. The possible values are:

Value	Meaning
O_normal	No rotation, not inverted
O_inverted	No rotation, inverted (reversed across horizontal midpoint)
O_r90	Rotated 90° clockwise
O_r90_in	Rotated 90° clockwise, then inverted
O_r180	Rotated 180°
O_r180_in	Rotated 180°, then inverted (this is mirror-reversed)
O_r270	Rotated 270° clockwise
O_r270_in	Rotated 270° clockwise, then inverted
O_w_on_b	Monochrome, negative image
O_bit_rev	Monochrome, reversed pixel order image

In order to reverse the visual orientation, to create an image which appears to be oriented correctly, you would supply the following values to the PICTools rotate opcode (OP\_ROR):

**Note: this is not available in AIMTools.**

Visual Orientation	Rotate
O_normal	O_normal
O_inverted	O_inverted
O_r90	O_r270
O_r90_in	O_r270_in
O_r180	O_r180
O_r180_in	O_r180_in
O_r270	O_r90
O_r270_in	O_r90_in

### CommentSize

Specifies the size of the buffer pointed to by the **Comment** field. This field is always input by the application.

If the application does not care to have image comments returned, then **CommentSize** is set to 0.

When an image format supporting comments is output from the operation, it is not necessary to set this field in order to specify the comment for the output image. **Comment** and **CommentLen** are the significant fields in that case.

When an image format supporting comments is input to the operation, the application has two ways to retrieve any image comment(s). The first way, when the application knows the largest size the comment can be or wants to limit the size which is returned, is to pre-allocate a comment buffer of the desired size. **CommentSize** is set to the size of the pre-allocated comment buffer. If a comment is encountered, and the length of the comment is larger than **CommentSize**, then only **CommentSize** bytes will be returned.

The second way, when the application wants to be certain it receives all comment data, it sets the **PF\_AllocateComment** flag in the **PicFlags** field in the operation structure. If **Pegasus** encounters a comment, it will return to the application with a RES\_ALLOCATE\_COMMENT\_BUF response. Pegasus will have set **CommentLen** to the required length for the comment. The application allocates a buffer of the required size, sets **CommentSize** to the allocated size, and continues the operation. If **Pegasus** encounters another comment it will return a RES\_HAVE\_COMMENT response. The application can save the previous comment's data as desired, and then continue the operation. **Pegasus** will immediately return a RES\_ALLOCATE\_COMMENT\_BUF response and the application can allocate a buffer exactly the size of the comment.

**Pegasus** stores and retrieves the comment as binary data. No trailing null terminator is added by **Pegasus**. Therefore, when the image may have been created by a different application, it is suggested that **CommentSize** be one byte larger than necessary to allow a trailing null terminator to be added by the application.

### **CommentLen**

Specifies the length of the comment in the image. This field is output by **Pegasus**. When **Pegasus** returns a RES\_ALLOCATE\_COMMENT\_BUF response, this field has been set to the length of the encountered comment, but the comment has not been copied to the comment buffer. When **Pegasus** returns a RES\_DONE or RES\_HAVE\_COMMENT response, this field has been set to the number of bytes returned for the last encountered comment. If **CommentSize** were not large enough for the entire comment to be returned, then **CommentLen** will be equal to the number of bytes copied, which will be equal to **CommentSize**.

If the comment buffer is large enough for the entire comment, then **CommentLen** is the exact size of the comment regarded as binary data. If the comment were stored with a null terminator, then **CommentLen** will include the null terminator. Otherwise, **CommentLen** will not include a null terminator.

### **Comment**

Points to a buffer for an image. This field is input by the application. For operations where the output image format supports comments, the application will copy the comment to be stored with the image to this buffer. For operations where the input image format supports comments, **Pegasus** will copy image comment(s) to this buffer. If **Comment** is not used to point to a buffer for a comment, it must be NULL.

### **App**

This field is reserved for the private use of the application. It provides a method for the application to include application-specific data in the PIC\_PARM structure. This field is input by the application if desired.

### **PercentDone**

Specifies the progress of the operation. The value will be between 0 and 100 and will only be 100 upon completion of the operation. This field is output by **Pegasus** for some operations.

### **Reserved**

This field is reserved for the private use of **Pegasus**. It must be set to 0 by the application before **Pegasus** is called with a request of REQ\_INIT. It will be set to 0 by **Pegasus** when **Pegasus** returns RES\_ERR and when **Pegasus** returns after it is called with a request of REQ\_TERM.

### **Get**

Points to a circular queue that holds input data for all **Pegasus** operations. This buffer is allocated and freed by the user. For additional information, see the description of the [QUEUE](#) structure and see the **Queue Management** section in the Programmer's Guide.

**Put**

Points to a circular queue to hold output data for all **Pegasus** operations. This buffer is allocated and freed by the user. For additional information, see the description of the [QUEUE](#) structure and see the **Queue Management** section in the Programmer's Guide.

**KeyField**

Specifies a password to use when compressing and decompressing the image. **KeyField** contains a password when the first byte is not 0. In this case, **KeyField** is regarded as 8 bytes of binary data and all 8 bytes are significant. If you use a password to protect a file, the image file will be quite difficult to decompress without the correct password. If decompressed by AIMTools and if the identical 8 binary bytes are not provided, Pegasus will return an error status of ERR\_INVALID\_KEY. If all 8 bytes of the KeyField field are 0, and the image requires a password, Pegasus will return an error status of ERR\_MISSING\_KEY. If a file does not require a password, it will be decompressed without error whether or not **KeyField** contains a password. This field is input by the application.

For some image formats, only the image data itself is protected by **KeyField** when some tool other than AIMTools is used to examine the image file. Comments, application data, etc. may not be protected by **KeyField** for these other tools.

This feature is supported only by these opcodes: OP\_D2S, OP\_D2SE, OP\_D2SEPLUS, OP\_S2D, OP\_SE2D, OP\_SE2DPLUS, OP\_S2P, OP\_LIP3, OP\_LIP3PLUS, OP\_LIE3PLUS, OP\_D2W and OP\_W2D.

**DeferFn**

Specifies a function to use for notifying the application of events which occur during an operation. Ordinarily, when an event occurs during an operation (e.g. RES\_GET\_NEED\_DATA), Pegasus returns to the application with the appropriate response code. For some CPU or O/S environments, this approach may be inconvenient or impractical. In those environments, **DeferFn** is used by **Pegasus** instead. The PIC\_PARM parameter is the same data structure which was passed to Pegasus. The RESPONSE parameter identifies the event. **DeferFn** is not called for RES\_ERR and RES\_DONE events. Pegasus must not be called from within the **DeferFn** function or any function the **DeferFn** calls. If the application wants to continue the operation, it returns 0. If the application wants to abort the operation, it returns a non-zero value. Note that when using the defer function from OP\_SCANFIX for RES\_QUERY calls, a non-zero return value will result in an output image not being returned; the **ScanFixReportData.Stop** value should be used for terminating the operation and returning the image in its current state.

**Flags**

<b>Value</b>	<b>Meaning</b>
F_Crop	For certain opcodes, if <b>F_Crop</b> is set, then <b>CropWidth</b> , <b>CropHeight</b> , <b>CropXoff</b> and <b>CropYOff</b> specify the cropping rectangle. Otherwise there is no cropping rectangle.
F_Raw	Set if REGION structure data is to be used for image parameters instead of <b>Head</b> .
F_UseDeferFn	When this flag is set and <b>DeferFn</b> points to a DeferFn callback function, then the opcode uses callback mode for returning responses other than RES_DONE and RES_ERR instead of using coroutine mode. See Appendix 2 for additional information on F_UseDeferFn.
F_UseAlternateCoroutine	This flag can be used when DeferFn is not used. If this flag is set on Windows, <b>Pegasus</b> will use a coroutine implementation which is based on Win32/x64 fibers. See Appendix 2 for additional information on F_UseAlternateCoroutine. This may be relevant if the client is also using fibers: <ul style="list-style-type: none"> <li>• If your application uses fibers in a thread calling Pegasus in alternate coroutine mode, convert the thread to a fiber before ever calling Pegasus in the thread and don't convert the fiber back to a thread until completely finished calling Pegasus in the thread.</li> <li>• If your application doesn't use fibers in a thread calling Pegasus in alternate coroutine mode, Pegasus will convert the thread to a fiber. In that case, if you call PegasusLibTerm in a different thread, but never having called PegasusLibThreadTerm in the thread that called Pegasus in alternate coroutine mode, then the dispatcher cannot convert the fiber back to a thread and a small amount of memory will be leaked.</li> <li>• If your application doesn't use fibers in a thread calling Pegasus in alternate coroutine mode, and if your application uses multiple dll's each linked to a static library dispatcher, and if the different dll's might each have active Pegasus operations in alternate coroutine mode in the same thread, then you can't safely call the PegasusLibThreadTerm function of any static library dispatcher until all Pegasus operations in alternate coroutine mode in the thread are finished.</li> </ul>

Any REQ\_CONT operation must be invoked in the context of the fiber under which the previous

	operation (REQ_INIT or REQ_EXEC) was invoked, regardless of whether the fiber was created by <b>Pegasus</b> or by the client. If this flag is set, and if the client application is multithreaded, any REQ_CONT operation must be invoked in the context of the thread under which the previous operation (REQ_INIT or REQ_EXEC) was invoked. New threads can not be used for REQ_CONT operations. If this flag is not set, the legacy coroutine implementation will be used.
F_UseStackSwitchCoroutine	Forces an internal stack-switch implementation for coroutines on a platform where that is not the default implementation. A stack-switch implementation may be marginally faster on some platforms, but is not as “well-behaved” from the point of view of the operating system which may rely on undocumented stack layout items for features such as exception or signal handling. See also F_UseAlternateCoroutine and F_UseDeferFn that override this flag. See Appendix 2 for additional information on F_UseStackSwitchCoroutine.
F_InputCrop	Same as F_Crop, but uses the larger IOCrop* fields in PIC_PARM allowing crop coordinates larger than 65535 to be specified. Cropping coordinates are with respect to the input image. Although they will be close to the same, cropping with F_InputCrop does not necessarily give the identical pixel values as cropping the input image before passing to the opcode or as cropping the output image after the opcode operation has output an uncropped image. This flag is currently supported only by the following opcodes: OP_CAD2D, OP_HDPHOTOE, OP_MDCA2D, OP_ZOOM2, OP_BINARIZE and OP_JLSE.
F_OutputCrop	Where an opcode supports it, this is the same as F_InputCrop, except cropping coordinates are with respect to the output image. Also cropping with F_OutputCrop gives the identical pixel values as cropping outside of the opcode after the opcode operation has output an uncropped image.

### Flags2

Reserved. The field should be set to 0 by the application.

**CropWidth**  
**CropHeight**  
**CropXoff**  
**CropYoff**

These fields are used to specify a cropping rectangle within an image for certain opcodes when **F\_Crop** is set in **Flags**.

### **ImageNumber**

This field is used to specify a particular image in a multi-image input file.

### **PacketType**

When RES\_EXTEND\_PIC2LIST response is returned, the type of the packet which was encountered and for which there was insufficient space is returned in **PacketType**. The application can look at the **PacketType** to decide whether or not to provide space for the packet.

### **SeekInfo**

Specifies an offset into the input or output stream which is required by the **Pegasus** operation. Its value is only defined after Pegasus has returned a **RES\_SEEK** response and until the operation is continued. This field is output by **Pegasus**. If the SEEK\_FILE bit of **SeekInfo** is clear, then a location in the input stream is specified. Otherwise a location in the output stream is specified. If the SEEK\_DIRECTION bit of **SeekInfo** is clear, then the desired offset is the **SeekInfo** offset regarded as the absolute offset from the beginning of the stream. Otherwise, the desired offset is computed by subtracting the **SeekInfo** offset from the offset of the end of the stream. In both cases, the **SeekInfo** offset is bits 0 through 29 of **SeekInfo**. It is most easily retrieved using ( **SeekInfo** & SEEK\_OFFSET ) to strip off the SEEK\_FILE and SEEK\_DIRECTION bits.

When seeking in the output stream, **Pegasus** will already have returned RES\_PUT\_NEED\_SPACE responses until the application emptied the output queue. The application should regard the next data output from the operation as beginning at the specified offset.

When seeking in the input stream, **Pegasus** will already have consumed all the data in the input queue, although it may not have emptied the queue. The application should set the queue to empty and should supply data to the queue beginning at the specified offset in response to the current RES\_SEEK or in response to RES\_GET\_NEED\_DATA.

### **PIC2List**

A pointer to a buffer to receive PIC2List information for opcodes supporting PIC2List data.

### **PIC2ListSize**

The size of the **PIC2List** buffer.

### **PIC2ListLen**

The length of the currently returned data in the **PIC2List** buffer.

### **RegionIn**

Reserved. The field should be set to 0 by the application.

### **RegionOut**

Reserved. The field should be set to 0 by the application.

**OpVersion0**

**OpVersion1**

**OpVersion2**

**OpVersion3**

64-bit FileVersion for opcode at REQ\_EXEC/RES\_DONE. This has the same values as the FileVersion in the opcode library's version resource.

**stkReserved**

Reserved for internal use

**opcodeExtraPtr**

Points to one of the \*\_EXTRA structures or 0. Before using, the \*\_EXTRA structure must be memset(&\_EXTRA, 0, sizeof(\_EXTRA)) and \_EXTRA.ExtraSize = sizeof(\_EXTRA)

**ReservedPtr3**

**ReservedPtr4**

**ReservedPtr5**

**ReservedPtr6**

**ReservedPtr7**

**ReservedPtr8**

Reserved. These fields should be set to 0 by the application.

**LoadPath**

This can be the explicit folder containing the opcode library. (see LoadResInstance discussion next.)

**LoadResInstance**

Module instance handle of an EXE or library from whose resource data the opcode library is to be loaded. The rules for using LoadPath and LoadResInstance are as follows:

if LoadPath == 0 && LoadResInstance == 0 then the opcode library is loaded from the same directory as the dispatcher library. If not found there then the opcode library is loaded using the directory order Windows uses when loading libraries

if LoadPath != 0 && LoadResInstance == 0, then the opcode library is loaded from the specified directory

if LoadPath == 0 && LoadResInstance != 0 then the opcode library is loaded from the specified module's resource data

if LoadPath != 0 && LoadResInstance != 0 then the opcode library is loaded from the specified module's resource data. If not found there, then the opcode library is loaded from the specified path. If LoadPath is "", and the opcode library is not found in the specified module's resource data, then the opcode library is loaded as though LoadPath == 0 && LoadResInstance == 0

**NestPP, ..., wrapReserved2**

Reserved. These fields should be set to 0 by the application.

**DispVersion0**  
**DispVersion1**  
**DispVersion2**  
**DispVersion3**

64-bit FileVersion for Dispatcher at REQ\_EXEC/RES\_DONE. This has the same value as the FileVersion in the dispatcher library's version resource.

**DispExports, ..., tlsReserved6**

Reserved. These fields should be set to 0 by the application.

**IOCropXoff**  
**IOCropYoff**  
**IOCropWidth**  
**IOCropHeight**

32-bit cropping coordinate fields for opcodes that support setting the F\_InputCrop flag or the F\_OutputCrop flag.

**NumberOfThreadsAllowed**

Certain opcodes can operate in a multithreaded manner. For those opcodes, this field specifies the number of threads to be used by the opcode (including the thread used to invoke the opcode.) A general rule of thumb is to set this field to the number of processors or processor cores available on the host hardware platform. To specify single threaded operation, set this field to zero or one.

The following opcodes support multithreading:

- OP\_J2KP
- OP\_J2KE
- OP\_J2KPRGB
- OP\_J2KERGB
- OP\_J2KP3D
- OP\_J2KE3D

**NumberOfThreadsUsed**

This is set by the opcode and indicates the number of threads actually used by the opcode (including the thread used to invoke the opcode.) For opcodes that do not support multithreading, this will be set to zero.

**SeekOffsetLow**  
**SeekOffsetHigh**

For opcodes that support seeking to past the 30-bit offsets offered by **SeekInfo**, these fields allow offsets up to 64-bits (for 32-bit platform opcodes) to be specified. (**SeekOffsetHigh** is always 0 on a 64-bit platform opcode). For an opcode that supports these, one or both of these fields will be set to a non-zero value and then both fields in combination specify the same seek offset for RES\_SEEK and RES\_POKE that **SeekInfo** specifies. However since **SeekInfo** uses bits 30 and 31 to specify which queue is seeking and whether seeking forward from the beginning or backward from the end, **SeekOffsetLow** with **SeekOffsetHigh** is the only way for an opcode to

specify offsets larger than 30 bits. **SeekInfo** bit 31 (SEEKINFO\_BACK\_FROM\_END) and **SeekInfo** bit 30 (SEEKINFO\_PUT\_QUEUE) should still be used to determine the queue and the direction of the offset. Opcode-independent code that also works on opcodes that don't yet support these fields, can be written by always looking at these fields when RES\_SEEK or RES\_POKE is returned. If either of these fields are non-zero, then these fields are used for the seek offset, otherwise **SeekInfo** & SEEK\_OFFSET is the seek offset.

**Reserveds[88]**

These fields should be set to 0 by the application.

**u**

Specifies a union of structures which hold information about a specific operation. The operation structure within the union is discriminated using the opcode is specified by the PIC\_PARM **Op** field. See the **Operation Structures** section for additional information. The structure used by each operation is:

<b>Opcode</b>	<b>Operation-Specific Structure</b>
OP_ADJUST	ADJUST
OP_BINARIZE	SB_PARMS
OP_CAD2D	OP_CAD
OP_CAMERARAW	CAMERARAW
OP_CLEAN	CLEAN
OP_D2F	D2F_STRUC
OP_D2FPLUS	D2F_STRUC
OP_D2J	DIB_INPUT
OP_D2PDF	PDF_UNION
OP_D2S	DIB_INPUT
OP_D2SE	DIB_INPUT
OP_EXP4	DIB_OUTPUT
OP_F2D	F2D_STRUC
OP_F2DPLUS	<a href="#">F2D_STRUC</a>
OP_HDPHOTO	HDP_UNION
OP_HDPHOTOE	HDP_UNION
OP_JBIG2E	JBIG2_UNION
OP_JBIG2P	JBIG2_UNION
OP_J2KE	<a href="#">J2K_UNION</a>
OP_J2KP	J2K_UNION
OP_J2KTRANSCODE	J2KT_UNION
OP_LIE3	LOSSLESS3
OP_LIE3PLUS	<a href="#">LOSSLESS3</a>
OP_LIP3	LOSSLESS3
OP_LIP3PLUS	LOSSLESS3
OP_PDF2D	PDF_UNION
OP_PNGP	PNG_UNION
OP_PNGE	PNG_UNION
OP_P2D	DIB_OUTPUT
OP_P2S	TRANP2S
OP_ROR	REORIENT
OP_S2D	DIB_OUTPUT
OP_SE2D	<a href="#">DIB_OUTPUT</a>
OP_S2P	TRANS2P
OP_UTL	UTL_STRUC
OP_ZOOM2	<a href="#">ZOOM_PARMS</a>
OP_W2D	WAVELET
OP_D2W	WAVELET

---

## QUEUE

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

A data structure specifying a circular queue. A **Pegasus** operation has an input or **Get**, queue and an output, or **Put**, queue. See the **Queue Management** section in the Programmer's Guide for additional information about using this structure with an ordinary linear buffer. See also Appendix 1 in this document for a description of how to use the input and output buffers as circular queues.

### Fields

#### FrontEnd

Reserved for **Pegasus**.

#### Start

Points to the first byte of the buffer being used for the queue.

#### Front

#### Rear

If **Front** equals **Rear**, then the queue is empty.

If the Q\_REVERSE flag is clear in **QFlags** then the queue is a forward queue and **Front** points to the next byte of valid data to be consumed. Valid data extends from **Front** forward (towards **End**) to the byte preceding **Rear**. If **Rear** is less than **Front**, then the queue data has wrapped around from **End** to **Start**.

If the Q\_REVERSE flag is set in **QFlags**, then the queue is a reversed queue and **Front** points to the first byte following the next byte of valid data to be consumed. Valid data extends from that next byte of valid data back (towards **Start**) to the byte pointed to by **Rear**. If **Rear** is greater than **Front**, then the queue data has wrapped around from **Start** to **End**.

#### End

Points to the first byte following the buffer being used for the queue.

#### RearEnd

Reserved for **Pegasus**

<b>QFlags</b>	
<b>Value</b>	<b>Meaning</b>
Q_EOF	Specifies that there is no more data. The application sets this bit in <b>QFlags</b> after the end of the data has been placed in the <b>Get</b> queue. This bit is input by the application.
Q_REVERSE	Causes the queue to operate in a back to front, or reversed, manner (see the <b>Queue Management</b> section in the Programmer's Guide for a more complete explanation of this flag). This bit is input by the application.
Q_IO_ERR	Set if an input or output error occurred. This bit is input by the application. It allows the application to terminate the operation at a low-level point where it may be more convenient to the application than calling <b>Pegasus</b> with <b>REQ_TERM</b> .
Q_READ_WRAP	Not used.
Q_DID_WRITE	Set by some opcodes (OP_LIE3 and OP_LIP3) if a RES_PUT_NEED_SPACE response has been returned by <b>Pegasus</b> since the <b>Put</b> queue was allocated. This bit is output by Pegasus.
Q_READING	Set at RES_PUT_NEED_SPACE if the data in the <b>Put</b> queue are not modified. In that case, and if the data are available at another location, the application may optionally discard the data instead of saving the data.

## RGBQUAD

```
typedef struct {  
    BYTE rgbBlue;  
    BYTE rgbGreen;  
    BYTE rgbRed;  
    BYTE rgbReserved;  
} RGBQUAD;
```

Each array element in the PIC\_PARM **ColorTable** is an RGBQUAD.

**rgbBlue**

**rgbGreen**

**rgbRed**

A value from 0..255 with the intensity of the corresponding color. 0 is the darkest (none of the color) and 255 is the most intense.

**rgbReserved**

This field is not used in **ColorTable**.

## Operations Structures

---

### F2D\_STRUC: OP\_F2D, OP\_F2DPLUS

```

typedef struct {
    DWORD           Reserved0;
    BYTE PICHUGE*   AuxSpace;
    BYTE PICHUGE*   Ptr2;
    BYTE PICHUGE*   GIFHead;
    DWORD PICHUGE*  TIFFIFDOffsetArray;
    BYTE PICHUGE*   MaskBuffer;
    BYTE PICHUGE*   Reserved6;
    BYTE PICHUGE*   Reserved7;
    BYTE PICHUGE*   Reserved8;
    PICFLAGS        PicFlags;
    PICFLAGS        PicFlags2;
    DWORD           AllocType;
    DWORD           AuxSize;
    DWORD           AuxUsed;
    DWORD           AuxNeeded;
    DWORD           ApplyResponse;
    DWORD           ProgressiveMode;
    BITMAPINFOHEADER BiOut;
    DWORD           YieldEvery;
    DWORD           PhotoCDResolution;
    LONG            Compression;
    BYTE            TransparentColorIndex;
    BYTE            RawG3FillOrder;
    BYTE            RawG3PhotometricInterpretation;
    BYTE            DisposalMethod;
    WORD            DelayTime;
    BYTE            BackgroundColor;
    BYTE            AspectRatio;
    DWORD           WidthPad;
    WORD            LogicalScreenWidth;
    WORD            LogicalScreenHeight;
    WORD            ImageLeftPosition;
    WORD            ImageTopPosition;
    DWORD           TIFFPhotometricInterpretation;
    DWORD           TIFFIFDOffset;
    BYTE            TIFFFirstByte;
    BYTE            OutBpp;
    BYTE            Expansion10c;
    BYTE            Expansion10d;
    DWORD           MaskBufferSize;
} F2D_STRUC;

```

The **F2D\_STRUC** structure supplies parameters to the [OP\\_F2D](#) and [OP\\_F2DPLUS opcodes](#) used to convert support image file formats (e.g., Targa, TIFF, etc.) to DIBs.

#### Fields

##### Reserved0

This field is not currently used and must be set to 0.

**AuxSpace**

**AuxSpace** may be set to point to a block of memory whose size is **AuxSize**. If applicable to the input image format and data, this block of memory will be returned by **OP\_F2D** organized as a sequential series of one or more "chunks". Each chunk consists of a DWORD **Tag**, followed by a DWORD **Length**, followed by **Length** bytes of data. Thus the total length of each chunk is **Length** + 8. **AuxUsed** is set by **OP\_F2D** to the sum of all the chunk(s) lengths output so far. See the [OP\\_F2D](#) input image format sections for additional information about **AuxSpace** and supported tags for each image format.

**Ptr2**

This field is not currently used by the **OP\_F2D** opcode and must be set to 0.

**GIFHead**

**GIFHead** may be set to point to a 781-byte block of memory in which **OP\_F2D** will return the GIF header. This is particularly useful if you want to process animated GIFs.

**TIFFIFDOffsetArray**

**TIFFIFDOffsetArray** may be set to point to an array of **PIC\_PARM.ImageNumber** DWORDs that **OP\_F2D** will fill with the TIFF IFD offsets from 1 to **PIC\_PARM.ImageNumber**.

**MaskBuffer**

**MaskBuffer** may be set to point to an array of size **MaskBufferSize** in which **OP\_F2D** will return the icon and mask. If zero, **OP\_F2D** will assign to **MaskBuffer** a pointer to a buffer it allocates (and frees during **REQ\_TERM**) to return the same information.

**Reserved6****Reserved7****Reserved8**

These fields are not used by the **OP\_F2D** opcode and must be 0.

**PicFlags**

Flags that control the **OP\_F2D** operation.

<b>Value</b>	<b>Meaning</b>
PF_IncludeBMPHead	If set, then the output data will begin with a BITMAPINFOHEADER, followed by a color map if applicable, followed by the DIB bits. If clear, then the output data will be the DIB bits only.
PF_IsGray	Set by <b>OP_F2D</b> on completion if the input image is a gray scale image. If a color table is present, and the color table consists entirely of shades of gray, then <b>PF_IsGray</b> is also set.

<b>Value</b>	<b>Meaning</b>
PF_ApplyTransparency	Set for an <b>OP_F2DPLUS</b> GIF image which has transparency data in order that transparent pixels are not set in the output DIB, allowing previous DIB pixels to show through the GIF image. If clear, then transparent pixels are set to the color whose color table index is the same as the transparent color index.
PF_NoDibPad	If set, <b>OP_F2D</b> will not pad the output DIB to a 4-byte boundary.
PF_WidthPadKnown	If set, the byte length of one DIB pixel line has been set by the application in <b>WidthPad</b> . Otherwise the length is computed by <b>OP_F2D</b> , padded to a 4-byte boundary unless <b>PF_NoDibPad</b> is set.

### PicFlags2

Flags that control the **OP\_F2D** operation.

<b>Value</b>	<b>Meaning</b>
PF2_IsInterlacedGIF	Set by <b>OP_F2D</b> on completion if the input image is an interlaced GIF.
PF2_GIFColorsSorted	Set by <b>OP_F2D</b> on completion if the input image is a GIF with sorted colors.
PF2_GIFAllow1and4Bpp	If set, <b>OP_F2D</b> will create 1 and 4 bpp DIBs for 2 and 16 color input GIFs, respectively. Otherwise, the output DIB will be 8 bpp.
PF2_PCXGetResolution	If set, PCX resolution information, if present, is biX/YPelsPerMeter.
PF2_RetrieveAlpha	If set, indicates that an alpha channel should be returned if any alpha information is present in the input file.

### AllocType

If non-zero, all input data must be present in the **Get** queue before **REQ\_EXEC**. **OP\_F2D** may advance the **Get.Front** pointer as it processes the input data. If so, the input data preceding **Get.Front** should not be considered to have been consumed by **OP\_F2D** and that data must not be modified. Even though the entire input image is present, **OP\_F2D** may issue **RES\_SEEK** events. If so, these events can be handled as:

```

case RES_SEEK:
    if ( ( PicParm.SeekInfo & SEEK_FILE ) == 0 )
    {
        PicParm.Get.Front =
            PicParm.Get.Start +
            ( PicParm.SeekInfo & SEEK_OFFSET );
    }
    else
    {
        /* if the Put queue is also large enough to hold
           the entire output image */
        PicParm.Put.Front = PicParm.Put.Rear =
            PicParm.Put.Start +
            ( PicParm.SeekInfo & SEEK_OFFSET );
    }

```

```

    }
    break;

```

If **AllocType** is 0, then a smaller input queue may be used which does not contain all the input image data. In that case, **OP\_F2D** may issue **RES\_SEEK** events to access non-sequential locations in the input image. Assuming that the input image data resides in an open file, these **RES\_SEEK** events might be handled as:

```

case RES_SEEK:
    if ( ( PicParm.SeekInfo & SEEK_FILE ) == 0 )
    {
        PicParm.Get.Front = PicParm.Get.Rear =
            PicParm.Get.Start;
        lseek(infile, PicParm.SeekInfo & SEEK_OFFSET, 0);
    }
    else
    {
        /* if the Put queue is also too small to hold
           entire output image */
        PicParm.Put.Front = PicParm.Put.Rear =
            PicParm.Put.Start;
        lseek(outfile, PicParm.SeekInfo & SEEK_OFFSET, 0);
    }
    break;

```

#### **AuxSize**

The size of the **AuxSpace** buffer.

#### **AuxUsed**

The total length in the **AuxSpace** buffer which is currently consumed by chunks.

#### **AuxNeeded**

The additional space needed in the **AuxSpace** buffer when a **RES\_AUX\_NEEDED** response is returned. If **AuxSize** is not increased by at least **AuxNeeded** when **REQ\_CONT** is called, then additional auxiliary data is not returned.

#### **ApplyResponse**

This field is not used by the **OP\_F2D** opcode and must be 0.

#### **ProgressiveMode**

This field is only used for **OP\_F2DPLUS** GIF images. Note that **ProgressiveModes** 1 and 3 are only useful if the data from prior interlace passes remain available in the **Put** queue for later interlace passes.

Value	Meaning
0	RES_DONE is returned four times. The first DIB returned is 1/8 the height of the full image. The second DIB returned is 1/4 the height of the full image. The third DIB returned is 1/2 the height of the full image. The final DIB returned is the complete image at full size.
1	RES_DONE is returned four times. Each returned DIB is the same size as the complete, full-size image. The first DIB has every 8 <sup>th</sup> line, the second DIB has every 4 <sup>th</sup> line and the third DIB has every 2 <sup>nd</sup> line. In each case, the lines between the interlace pass lines are filled with copies of the preceding interlace pass line. The final DIB is the complete image at full size.
2	RES_DONE is returned once. The returned DIB is the complete image at full size.
3	RES_DONE is returned four times. Each returned DIB is the same size as the complete, full-size image. The first DIB has every 8 <sup>th</sup> line, the second DIB has every 4 <sup>th</sup> line, the third DIB has every 2 <sup>nd</sup> line and the fourth DIB is the complete image. The first DIB fills in the lines between each 8 <sup>th</sup> line with copies of the preceding line. The second DIB adds image data from the second interlace pass to the first DIB. The third DIB adds image data from the third interlace pass to the second DIB. The final DIB adds image data from the fourth interface pass to the third DIB. Only the first DIB fills in the lines between the current interface pass lines.

### BiOut

When **OP\_F2D** is complete (RES\_DONE after REQ\_EXEC), then **BiOut** contains the BITMAPINFOHEADER for the output DIB, whether or not **PF\_IncludeBMPHeader** has been set in **F2D.PicFlags**.

### YieldEvery

If **PF\_YieldPut** is set in **F2D.PicFlags**, then **OP\_F2D** returns a **RES\_PUT\_DATA\_YIELD** response after every **YieldEvery** lines have been output to the **Put** queue.

### PhotoCDResolution

This field is not currently used by the **OP\_F2D** opcode and must be 0.

### Compression

The TIFF input image compression is returned in this field.

Value	Meaning
0	Uncompressed
1	Uncompressed
2	Modified G3
3	1-dimensional G3
-3	2-dimensional G3
4	G4
5	LZW
6	JPEG (old style)
7	JPEG (new style)
8	Deflate (ZLIB)
32946	Deflate (ZLIB)
32773	Packbits

### TransparentColorIndex

For **OP\_F2DPLUS**, if a GIF image has transparent pixels, then when **OP\_F2DPLUS** is complete, this value is the color index of the transparent pixels. If **PF\_ApplyTransparency** were set, then the output DIB will not have replaced DIB pixels corresponding to GIF image pixels with this color index so this value will not be useful. If **PF\_ApplyTransparency** is not set, then pixels in the output DIB which have this color index value were intended to be transparent pixels.

#### **RawG3FillOrder**

If a raw G3 fax is input, this field allows the TIFF FillOrder to be specified. **RawG3FillOrder** equal to 1 is equivalent to the usual TIFF FillOrder equal to 1, filling bytes from most-significant-bit to least-significant-bit. **RawG3FillOrder** not equal to 1 is equivalent to the unusual TIFF FillOrder equal to 2, filling bytes from lsb to msb. Although the usual *TIFF* FillOrder is 1, raw G3 faxes will usually require a **RawG3FillOrder** of 2.

#### **RawG3PhotometricInterpretation**

If a raw G3 fax is output, this field allows the TIFF PhotometricInterpretation to be specified. If **RawG3PhotometricInterpretation** is 0, then the input G3 fax is interpreted as the usual TIFF WhiteIsZero. Otherwise, the input G3 fax is interpreted as the unusual TIFF BlackIsZero. Ordinarily, raw G3 faxes will require a **RawG3PhotometricInterpretation** of 0.

#### **DisposalMethod**

For a GIF image, if the GIF89a Graphics Extension block is present, the disposal method field will be returned by **OP\_F2D** in **DisposalMethod**. This field indicates what should occur after the graphic is displayed.

#### **DelayTime)**

For a GIF image, if the GIF89a Graphics Extension block is present, the delay time field will be returned by **OP\_F2D** in **DelayTime**. This field specifies the time (in hundredths of a second) to wait before continuing processing the data stream.

#### **BackgroundColor**

For a GIF image, the background color from the GIF header will be returned by **OP\_F2D** in **BackgroundColor**.

#### **AspectRatio**

For a GIF image, the pixel aspect ratio from the GIF header will be returned by **OP\_F2D** in **AspectRatio**. The actual pixel aspect ratio can be computed with

$$\text{PixelAspectRatio} = \text{PixelWidth}/\text{PixelHeight} = (\mathbf{AspectRatio} + 15) / 64$$

#### **WidthPad**

Specifies the width of an uncompressed DIB pixel line in bytes. **WidthPad** takes into account the number of bits needed to represent a pixel and any padding that may be required at the end of each line. **WidthPad** will be output by **Pegasus** aligned to a 4-byte boundary unless the **PF\_WidthPadKnown** flag is set in **PicFlags**. If **PF\_WidthPadKnown** is set, then **WidthPad** must be input by the application. If **PF\_NoDIBPad** is set, then the **WidthPad** computed by **Pegasus** will not include any padding necessary to align the DIB to a 4-byte boundary.

#### **LogicalScreenWidth**

For a GIF image, the logical screen width from the GIF header will be returned by **OP\_F2D** in **LogicalScreenWidth**.

#### **LogicalScreenHeight**

For a GIF image, the logical screen height from the GIF header will be returned by **OP\_F2D** in **LogicalScreenHeight**.

#### **ImageLeftPosition**

For a GIF image, the image left position (the column number of the left edge of the image) from the GIF image descriptor will be returned by **OP\_F2D** in **ImageLeftPosition**.

#### **ImageTopPosition**

For a GIF image, the image top position (the row number of the top edge of the image) from the GIF image descriptor will be returned by **OP\_F2D** in **ImageTopPosition**.

#### **TIFFPhotometricInterpretation**

For a TIFF image, the value of the PhotoMetricInterpretation tag will be returned by **OP\_F2D** in **TIFFPhotometricInterpretation**.

<b>Value</b>	<b>Meaning</b>
0	For bilevel and grayscale images 0 is imaged as white.
1	For bilevel and grayscale images 0 is imaged as black.
2	RGB 24bpp, no palette.
3	RGB palette.
5	CMYK.

#### **TIFFIFDOffset**

If **TIFFFirstByte** is 'I' or 'M', then **TIFFIFDOffset** should be set either to the offset of the IFD to start scanning for the **PIC\_PARM.ImageNumber** image, or to 0 to start scanning at the first IFD. Upon return, **OP\_F2D** will set this member to the offset of the IFD for the requested image.

#### **TIFFFirstByte**

If **TIFFFirstByte** is 'I' or 'M' and **TIFFIFDOffset** is set, **OP\_F2D** will skip reading the TIFF header.

#### **OutBpp**

For a BMP, ICO or TARGA image, if **OutBpp** is 32 and the input is 32 bpp, the output will be 32 bpp as well.

#### **Expansion10c**

#### **Expansion10d**

These fields are not used by the **OP\_F2D** opcode and must be 0.

#### **MaskBufferSize**

For an ICO image, the number of bytes in MaskBuffer.

---

## LOSSLESS3: OP\_LIE3, OP\_LIE3PLUS

```

typedef struct {
    DWORD           Reserved0;
    LAYER PICHUGE*  Layer;
    BYTE PICHUGE*   AHT;
    BYTE PICHUGE*   Reserved3;
    RGBQUAD PICHUGE* UniversalCT;
    BYTE PICHUGE*   Reserved5;
    BYTE PICHUGE*   Reserved6;
    BYTE PICHUGE*   Reserved7;
    BYTE PICHUGE*   Reserved8;
    PICFLAGS        PicFlags;
    PICFLAGS        PicFlags2;
    REGION          Region;
    LONG            StripSize;
    SHORT           NumUC;
    SBYTE           MinLayerWanted;
    SBYTE           CurrentLayer;
    BYTE            RestartLo;
    BYTE            RestartHi;
    BYTE            ProgHeight;
    BYTE            NumRestarts;
    DWORD           RestartOff[4];
    SHORT           NumOfPages;
    SHORT           PageNum;
    SHORT           Transparent;
    SHORT           UserDelay;
    SHORT           XOff;
    SHORT           YOff;
    BYTE            DispMethod;
    BYTE            AllowedBitErr;
    BYTE            CompMethod;
    BYTE            CompOrder;
    BYTE            PTuning;
    BYTE            Channel;
    BYTE            IOPixType;
    BYTE            NativeBpp;
    DWORD           RestartCount;
    DWORD           SetBits;
    DWORD PICHUGE*  PixelMap;
    BYTE            EffectiveBpp;
    BYTE            PrecisionUCT;
    BYTE            Reserved9[2];
    DWORD           AHTLen;
    DWORD           AppsToOmit ;
    DWORD           NumUC2;
    LONG            ImageWidth;
    LONG            ImageHeight;
} LOSSLESS3;

```

The **LOSSLESS3** structure supplies parameters to the [OP\\_LIE3](#) opcode used for expanding IMStar or lossless JPEG images to DIBs.

**Fields**

**Reserved0**

These fields are not currently used by **OP\_LIE3** and must be set to 0.

**Layer**

**AHT**

**Reserved3**

These fields are not currently used by **OP\_LIE3** and must be set to NULL.

**UniversalCT**

For lossless JPEG images **UniversalCT** allows the application to specify a translation from input pixel gray scale or color component brightness levels to corresponding output pixel values. The number of elements in the translation array is **u.LL3.NumUC**.

For gray scale lossless JPEG images **UniversalCT** allow the application to specify a translation from input pixel gray levels into output pixel gray levels.

If there are at least as many **UniversalCT** elements as input gray levels, then each **UniversalCT** element contains the 8-bit output gray level for each input pixel whose gray level is the same as the 0-based index of the **UniversalCT** element.

If there are fewer **UniversalCT** elements than input gray levels, then each **UniversalCT** element contains the 8-bit output gray level for all input pixels whose gray level, when scaled to the number of elements in **UniversalCT**, results in the 0-based index of the **UniversalCT** element. The input gray level is scaled to the number of elements in **UniversalCT** as:

$$\text{UniversalCTIndex} = (\text{InputGrayLevel} * \text{NumGrayLevels} + \text{NumGrayLevels} / 2) / \text{NumUC}$$

**Reserved5**

**Reserved6**

**Reserved7**

**Reserved8**

These fields are not currently used by **OP\_LIE3** and must be set to NULL.

**PicFlags**

Flags that control or report on the **OP\_LIE3** operation.

<b>Value</b>	<b>Meaning</b>
PF_HaveWatermark	If set by <b>OP_LIE3</b> , then a watermark was encountered in the input image and returned in <b>PIC2List</b> . If the <b>PIC2List</b> is 0 or <b>PIC2ListSize</b> is 0, or if the <b>PIC2List</b> is not reallocated in response to a <b>RES_EXTEND_PIC2LIST</b> response, then the watermark may be discarded and <b>PF_Watermark</b> is not set.
PF_IsTransparency	If set by <b>OP_LIE3</b> , then <b>u.LL3.Transparent</b> contains the color index for transparent pixels. Otherwise <b>u.LL3.Transparent</b> is -1.
PF_YieldPut	If set by the application, then <b>OP_LIE3</b> will periodically return <b>RES_PUT_DATA_YIELD</b> as it expands the image.
PF_IgnoreBadSuffix	Force the decompressor bug allowing non-compliant images to be decompressed. If <b>PF_AutoIgnoreBadSuffix</b> is set requesting automatic detection and decoding, then <b>OP_LIE3</b> sets or clears this flag to indicate whether it used non-compliant or compliant decoding, respectively.
PF_AutoIgnoreBadSuffix	Use the proprietary automatic detection algorithm to decode a 16 bpp image that may or may not be compliant.

**PicFlags2**

Additional Flags that control or report on the **OP\_LIE3** operation.

<b>Value</b>	<b>Meaning</b>
PF2_FastDecompress	The more optimized Lossless JPEG decompressor was used because it was 8bpp grayscale, predictor 1, with a full size Put queue, with an appropriate Huffman table.
PF2_FoundDicom	Set by the opcode to signify that there may be more images in the GET buffer.
PF2_FoundEOI	Set by the opcode. Means that the End Of Image marker was reached

**Region**

This field is used to define image characteristics. **OP\_LIE3** returns the image's characteristics in this structure in addition to returning them in **PIC\_PARM.Head**. For images with **Stride** greater than 32767, **REGION2** should be used (see the discussion of the [REGION2](#) structure) and **ParmVerMinor** must be at least 4.

**StripSize**

Following **REQ\_INIT**, this is set to the size of one input strip. The **Put** queue buffer must be large enough to hold at least one strip.

### **NumUC**

**NumUC** contains the size of **UniversalCT** in bytes.

### **MinLayerWanted**

### **CurrentLayer**

### **RestartLo**

### **RestartHi**

### **ProgHeight**

### **NumRestarts**

### **RestartOff**

These fields are not used by **OP\_LIE3** and must be set to NULL.

### **NumOfPages**

Retrieved from the image by **OP\_LIE3**, but otherwise ignored. This can be regarded as a comment about the image which is intended to hold the total number of pages or images in some arbitrary group of images which includes the current image.

### **PageNum**

Retrieved from the image by **OP\_LIE3**, but otherwise ignored. This can be regarded as a comment about the image which is intended to hold the page number or image number of the current image in some arbitrary group of images which includes the current image.

### **Transparent**

Retrieved from the image by **OP\_LIE3**, but otherwise ignored. If **PF\_IsTransparency** in **u.LL3.PicFlags** is set by **OP\_LIE3**, then this value was retrieved from the image. It is intended to denote a transparent color index.

An application might treat image pixels which have that color index as transparent pixels. If the application were to overlay the current image onto another image, then the corresponding pixels in the overlaid image would not be replaced by transparent pixels in the current image.

### **UserDelay**

Retrieved from the image by **OP\_LIE3**, but otherwise ignored. This value is intended to represent the number of .01 second intervals to wait before replacing the current image with the next image.

### **Xoff**

Retrieved from the image by **OP\_LIE3**, but otherwise ignored. If the application were to overlay the current image onto another image, then this value would represent the intended horizontal offset of the current image from the left edge of the overlaid image.

### **Yoff**

Retrieved from the image by **OP\_LIE3**, but otherwise ignored. If the application were to overlay the current image onto another image, then this value would represent the intended vertical offset of the current image from the top edge of the overlaid image.

### **DispMethod**

Retrieved from the image by **OP\_LIE3**, but otherwise ignored. If the application were presenting a series of these images, then this value might represent the transition method to be used for replacing the current image with the next image.

### **AllowedBitErr**

Retrieved from the image by **OP\_LIE3**. If **AllowedBitError** is not 0, it specifies the number of least significant bits of input that were truncated from the gray level (or from each of the RGB levels for 24-bit RGB). In this case, the compression is not lossless compression; the source image will not be exactly reconstructed from the compressed image.

### **CompMethod**

Retrieved from the image by **OP\_LIE3**. The compression methods supported by the LIP3 opcodes are: lossless JPEG compression (METHOD\_JPEG), lossless LOCO compression (METHOD\_LOCO), and lossless IMPACSTAR compression (METHOD\_PPMD).

### **CompOrder**

Retrieved from the image by **OP\_LIE3**. If **CompMethod** is METHOD\_JPEG, then **CompOrder** specifies the Predictor value used by lossless JPEG. Allowed values are in the range 1 – 7. If **CompMethod** is METHOD\_PPMD, then **CompOrder** is the order, or length, of the context which is to be used to predict the image pixels.

### **PTuning**

Retrieved from the image by **OP\_LIE3**.

### **Channel**

Color channel to be treated as gray. Not used.

### **IOPixType**

Convert to this Pixel Type.

### **NativeBpp**

Native Bpp prior to any conversion from/to IOPixType

### **RestartCount**

Number of pixels between restart markers

### **PrecisionUCT**

Sets the precision of the translation mapping. Each value in the translation table is less than  $2^{\text{PrecisionUCT}}$ .

### **AHTLen**

Length in bytes of the Alternative Huffman Table

**AppsToOmit**

This field is not used by **OP\_LIE3**.

---

## J2K\_EXTRA: OP\_J2KE, OP\_J2KE3D, OP\_J2KERGB, OP\_J2KP, OP\_J2KP3D, OP\_J2KPRGB

```
typedef struct {  
    DWORD          ExtraSize;  
    double         CaptureResVert;  
    double         CaptureResHorz;  
    double         DisplayResVert;  
    double         DisplayResHorz;  
} J2K_EXTRA;
```

This structure must be zeroed and have ExtraSize set to sizeof(J2K\_EXTRA) before use. To use this structure set PIC\_PARM.opcodeExtraPtr to point to a valid J2K\_EXTRA structure. If PIC\_PARM.opcodeExtraPtr does not point to a J2K\_EXTRA structure no information will be returned via opcodeExtraPtr.

### Fields

#### **ExtraSize**

Must be the size of J2K\_EXTRA in bytes.

#### **CaptureResVert**

Vertical Capture resolution in canvas grid points per meter.

#### **CaptureResHorz**

Horizontal Capture resolution in canvas grid points per meter.

#### **DisplayResVert**

Vertical Display resolution in canvas grid points per meter.

#### **DisplayResHorz**

Horizontal Display resolution in canvas grid points per meter.

---

## J2K\_UNION: OP J2KE, OP J2KE3D, OP J2KERGB, OP J2KP, OP J2KP3D, OP J2KPRGB

```

typedef struct {
    DWORD           Reserved0;
    BYTE PICHUGE*   UpdatedTiles;
    BYTE PICHUGE*   ColorSpecICCProfileData;
    BYTE PICHUGE*   Reserved3;
    BYTE PICHUGE*   Reserved4;
    BYTE PICHUGE*   Reserved5;
    BYTE PICHUGE*   Reserved6;
    BYTE PICHUGE*   Reserved7;
    BYTE PICHUGE*   Reserved8;
    PICFLAGS        PicFlags;
    PICFLAGS        PicFlags2
    /* following in reference grid coordinates */
    DWORD           ImageWidth;
    DWORD           ImageHeight;
    DWORD           ImageXOff;
    DWORD           ImageYOff;
    DWORD           StripSize;
    REGION          Region;
    DWORD           NumOtherPartitions;
    PARTITION PICHUGE* OtherPartitions;
    DWORD           PartitionNum;
    DWORD           TileWidth;
    DWORD           TileHeight;
    DWORD           TileXOff;
    DWORD           TileYOff;
    TILE            DftTile;
    DWORD           NumOtherTiles;
    TILE PICHUGE*   OtherTiles;
    DWORD           Rate;
    THUMBNAIL       Thumbnail;
    DWORD           Resolution;
    DWORD           CompFileSize;
    DWORD           NumPartitions;
    DWORD           ExpandLayers;
    double          TargetPSNR;
    DWORD           ExpandResolution;
    DWORD           SliceOff;
    DWORD           JPEG2000FileFormat;
    DWORD           JPEG2000Profile;
    BYTE            ColorSpecMethod;
    SBYTE           ColorSpecPrecedence;
    BYTE            ColorSpecApproximation;
    BYTE            ColorSpecEnumeratedColorSpace;
    DWORD           ColorSpecICCProfileLen;
    DWORD           CompFileSizeHigh;
} J2K_UNION;

```

**Fields****Reserved 0**

Must be 0

### UpdatedTiles

Ordinarily this is set by [OP\\_JPIPCLIENT](#) or OP\_RIDP2. UpdatedTiles is a bitmap of updated 64x64 pixel subrectangles. All subrectangles denoted by a '1' bit are decompressed. If UpdatedTiles <> 0, then full image size output buffers are required. UpdatedTiles are disallowed in reduced-memory mode of the decompressor. For an image width W, the UpdatedTiles bit corresponding to pixel (x, y) is:

```
int bitsperrow = ( W + 63 ) / 64;
int yofs = y / 64;
int xyofs = ( x / 64 ) + ( bitsperrow * yofs );
int xybyteofs = xyofs / 8;
int xybitofs = 1 << ( xyofs % 8 );
if ( UpdatedTiles[xybyteofs] & xybitofs ) != 0 );
// bit is set
```

### ColorSpecICCPProfileData

Pointer to a valid ICC Profile.

### Reserved3 thru 8

Must be NULL.

### PICFlags

Value	Meaning
PF_YieldPut	(OP_J2KE) Set if RES_PUT_DATA_YIELD response is desired for example for progress reporting during image decompression.
PF_YieldGet	(OP_J2KP) Set if RES_GET_DATA_YIELD response is desired for example for progress reporting during image compression.

### PICFlags2

Value	Meaning
PF2_SkipJP2Header	(OP_J2KP, OP_J2KE) If set with OP_J2KP, the Put buffer contains the raw JPEG2000 codestream, without the JP2 or JPX file header. Not recommended. If raw JPEG2000 codestream, without file header is encountered OP_J2KE sets this flag.
PF2_ForceLossless	(OP_J2KP only) Compress the image losslessly, using the reversible color-space and wavelet transforms. Overrides Rate, CompFileSize, TileCompTransform, and WaveletTransform.
PF2_KeepColorTable	For Color Mapped files only, when RF_CM2RGB is specified, optionally keep the color table with the compressed image returned by OP_J2KP or return the color table along with the decompressed image returned by OP_J2KE. This option is redundant when a Color Mapped image and RF_CM2RGB is not specified. This option is ignored if PF2_SkipJP2Header is set in OP_J2KP.

PF2_SkipUpsampling	(OP_J2KE only) Don't upsample the image, leave components downsampled
PF2_AlternateRateDistortion	(OP_J2KP only) not implemented.
PF2_Use32BitPrecision	Process samples during compression and decompression with no more than 8 bits per sample precision using a full 32-bits of precision instead of 16-bits of precision. Performance is slower when this flag is specified, but compression/decompression is very slightly more accurate for lossy compression.
PF2_InsertTLMMarker	(OP_J2KP only) Insert TLM marker, adding J2K header data that is an array of all tile part lengths.
PF2_InsertPPMMarker	(OP_J2KP only) Insert PPM marker facilitating random-access to improve JPIP server stateless performance.
PF2_EarlyOutRateControl	(OP_J2KP only) If set, compression is faster for more compression than for less compression but for high-gray the compressor may not hit the desired rate as closely for some images at highest quality settings.
PF2_FavorSpeed	(OP_J2KP only) Use fastest coding mode for encoding; files created using this flag will also decode faster. Note that when this flag is set, lossless encoded files may become slightly larger and lossy rate-controlled encoded files may decode to slightly lower quality. Can be combined with PF2_EarlyOutRateControl for best performance.
PF2_UseResSeek	(Win32/x64) Allow RES_SEEK events allowing seeking in the input. When Thumbnail, ExpandResolution, or Resolution are set or when cropping, this allows expanding without reading all data in the input file. In the reduced-memory mode of OP_J2KE, this flag is used to choose between a mode that requires seeking in the input and one that operates without any seeks. If the compressed image contains more than one tile, the reduced-memory mode requires that this flag be set and seeking in the input be allowed; otherwise, this flag should be clear for best reduced-memory mode performance. In the reduced-memory mode of OP_J2KP when the application requests that more than one tile be created, this flag must be set and seeking must be allowed in the compressed output stream.
PF2_MMXDisable	Set to disable MMX and later optimizations.
PF2_P2AndP3OptDisable	Set to disable P2, P3 and later optimizations.
PF2_P3OptDisable	If the CPU supports SSE instructions, but the O/S doesn't support SSE instructions, then you must set PF2_P3OptDisable or the CPU will fault.
PF2_P4OptDisable	If the CPU supports SSE2 instructions, but the O/S doesn't support SSE2 instructions, then you must set PF2_P4OptDisable or the CPU will fault.

PF2_ARMAsmDisable	Set to disable use of ARM processor optimizations.
PF2_ARMNeonDisable	Set to disable use of ARM advanced SIMD (NEON) processor optimizations.
PF2_3D_Slices	(J2K volumetric only) Set to instruct opcode to encode input partitions as 3-D slices (one slice per partition) instead of as 2-D components. Returned by opcode to indicate that JPEG2000 image is a 3-D slice encoding and a slice will be returned in each partition.
PF2_OmitPICMetaData	(OP_J2KP only) Set to suppress inserting Pegasus informational metadata (UUID box) in the output image. It is recommended that this flag not be set.
PF2_Reduced-MemoryMode	Set to select the reduced-memory mode of operation

The following four elements describe the size and position of the image on the reference grid or canvas. JPEG 2000 allows the upper-left corner to be assigned an arbitrary nonnegative position on a background reference grid. The positions of the other three corners are then constrained by this corner's position and the image dimensions.

**ImageWidth**

This is the width of the image in pixels

**ImageHeight**

This is the height of the image in pixels

**ImageXOff**

X-position of the image upper-left corner on the JPEG 2000 reference grid. A value of 0 is recommended unless there are specific reasons to do otherwise. The value must be nonnegative.

**ImageYOff**

Y-position of the image upper-left corner on the JPEG 2000 reference grid. A value of 0 is recommended unless there are specific reasons to do otherwise. The value must be nonnegative

**StripSize**

This is the minimum size of the buffer to hold data in the first partition. For OP\_J2KP, this is the minimum size of the Get Buffer. For OP\_J2KE, this is the minimum size of the Put buffer. This is provided to the caller on return from the REQ\_INIT phase.

**Region**

This is the region for the first partition. See Region below. For images with **Width, Height, or Stride** too large to fit in **REGION**, **REGION2** should be used (see the discussion of the [REGION2](#) structure) and **ParmVerMinor** must be at least 4.

The following three elements describe the partition structure for the uncompressed/decompressed image data.

### **NumOtherPartitions**

The number of OtherPartitions when there is more than one. This field is ignored when OtherPartitions is NULL. The PicParm.u.J2K.StripSize, PicParm.u.J2K.Region, and PicParm.Get or PicParm.Put queues specify the first partition's components explicitly, and this field specifies all other partitions' components. For OP\_J2KP, the maximum allowed value for this field is 16383, because the JPEG2000 standard allows a maximum of 16384 components. For OP\_J2KE, if the actual number of partitions required to decode the image is less than the number provided by the app, then the value of this field and the extraneous partitions are ignored and just the required number of partitions are decoded and returned. If the application provides fewer partitions than are required by the image, then just the partitions provided are decoded and returned.

### **OtherPartitions**

This is a pointer to an array of NumOtherPartition partitions, one for each partition after the first, making up the image. If this field is NULL, then only one partition is encoded or decoded. See Partition below.

### **PartitionNum**

This is the number designating which partition is the one to be used whenever a response code RES\_GET\_NEED\_DATA or RES\_PUT\_NEED\_SPACE is returned to the application. If the value is 0, for the first partition, then the PicParm.Get or PicParm.Put queue is being used along with the u.J2K.Region information. Otherwise the value indicates that the Get, Put, and Region structures in the designated partition are to be used for buffer manipulation etc.

The following four elements describe the size and position of the tile grid. This grid can be positioned arbitrarily with respect to both the image and the reference grid.

### **TileWidth**

This is the width of the tile. If 0 (recommended in most cases) then the tile width = ImageWidth

### **TileHeight**

This is the height of the tile. If 0 (recommended in most cases) then the tile height = ImageHeight

### **TileXOff**

X-position of the upper-left corner of the upper-leftmost tile on the JPEG 2000 reference grid. A value of 0 is recommended unless there are specific reasons to do otherwise. The value must be nonnegative and it is constrained to be greater than ImageXOff – TileWidth, but no greater than ImageXOff

### **TileYOff**

Y-position of the upper-left corner of the upper-leftmost tile on the JPEG 2000 reference grid. A value of 0 is recommended unless there are specific reasons to do otherwise. The value must be nonnegative and it is constrained to be greater than ImageYOff – TileHeight, but no greater than ImageYOff..

### **DftTile**

Compression settings for all tiles. When implemented, exceptions for individual tiles can be specified in the OtherTiles array. See [TILE](#).

### **NumOtherTiles**

Not implemented. The number of tiles in the OtherTiles array.

### **OtherTiles**

Not implemented. An array of special compression settings for particular tiles. These settings allow the u.J2K.DftTile settings to be overridden for particular tiles.

### **Rate**

(OP\_J2KP) The number of thousandths of a bit per pixel in the compressed image—used to control degree of compression; overrides custom compression settings if set. If set to 0 then the degree of compression is set by CompFileSize.

### **Thumbnail**

(OP\_J2KE) power of 2 by which to scale down size. Allows for faster decompression to a smaller image size and requires less compressed image data depending on the progression ordering.

### **Resolution**

(Solaris, AIX only) (OP\_J2KE) A full-size image is returned but the Resolution number highest-detail bitslice layers of the image are skipped resulting in a fuzzier image, but requiring less compressed image data depending on the progression ordering.

### **CompFileSize**

(OP\_J2KP) Desired compressed file size in bytes; overrides Rate or custom compression settings if set. If 0 then the Rate field is used to determine compression size. If Rate is also 0, then the compressed file size is not arbitrarily limited in size but will depend on the compressibility of the image and on the other compression settings.

### **NumPartitions**

(OP\_J2KE) After RES\_DONE is returned from REQ\_INIT, NumPartitions minus one is the number of OtherPartitions required to decompress all image components. If NumOtherPartitions < NumPartitions – 1, then the application could decompress all components by calling REQ\_TERM to terminate the current OP\_J2KE operation and could then start a new OP\_J2KE operation on the same image after reallocating the OtherPartitions array and setting NumOtherPartitions. If REQ\_EXEC is called and NumOtherPartitions < NumPartitions – 1, then the extra components aren't decompressed.

### ExpandLayers

(OP\_J2KE) A full-size image is returned but the ExpandLayers number highest-detail bitslice layers of the image are skipped resulting in a fuzzier image, but requiring less compressed image data depending on the progression ordering.

### TargetPSNR

(OP\_J2KP) Set this to the desired compressed image quality as measured by Peak-Signal-to-Noise-Ratio.

### ExpandResolution

(OP\_J2KE) A full-size image is returned but the ExpandResolution number highest-detail wavelet decomposition levels of the image are skipped resulting in a fuzzier image, but requiring less compressed image data depending on the progression ordering.

### SliceOff

(J2K volumetric expand only) Set this to the offset of the first slice to expand. The number of slices to expand, starting from this offset, is determined by the number of partitions provided to the opcode.

### JPEG2000FileFormat

This field reports the JPEG2000 file format of the input JPEG2000 image. It is informational only and is returned by REQ\_INIT.

Value	Meaning
JPEG2000FileFormat_J2K	Input is a JPEG2000 raw codestream image.
JPEG2000FileFormat_JP2	Input is a JPEG2000 JP2 file format image.
JPEG2000FileFormat_JPX	Input is a JPEG2000 JPX file format image.

### JPEG2000Profile

This field reports the JPEG2000 profile as determined from indicators in the JPEG2000 image. It is informational only and is returned by REQ\_INIT for the expand opcodes. It can be set before REQ\_INIT for the pack opcodes when creating a JPEG2000 file. See the JPEG2000 ISO standard for more information about these profiles.

Value	Meaning
JPEG2000Profile_None	No profile – no restrictions (Part-1)
JPEG2000Profile_0	Profile-0 – most restrictive (Part-1)
JPEG2000Profile_1	Profile-1 – some restrictions (Part-1)
JPEG2000Profile_2	Same as JPEG2000Profile_None
JPEG2000Profile_Part2_Full	Requires full capabilities of JPEG2000 Part-2
JPEG2000Profile_Part2_JPXBaseline	Conforms to JPEG2000 JPX baseline (Part-2)

### ColorSpecEnumeratedColorSpace

This field reports the Enumerated Color Space if ColorSpecMethod == J2K\_COLOR\_SPEC\_METHOD\_ECS. It is returned during REQ\_INIT for the expand opcodes and is not used for the pack opcodes.

<b>Value</b>	<b>Meaning</b>
J2K_COLOR_SPEC_ECS_RGB	sRGB images as defined by IEC 61966-2-1 and ISO/IEC 15444-1:2004.
J2K_COLOR_SPEC_ECS_GREYSCALE	Greyscale images as defined by IEC 61966-2-1 and ISO/IEC 15444-1:2004.
J2K_COLOR_SPEC_ECS_YCC	sYCC images as defined by IEC 61966-2-1 Amd. 1 and ISO/IEC 15444-1:2004.
0	Other.

**ColorSpecMethod**

This field reports the JPEG2000 Color Specification Method as determined from the Color Specification Box in the JP2 Header box. It is returned by REQ\_INIT for the expand opcodes. It can be set, only to J2K\_COLOR\_SPEC\_METHOD\_ICP, for the pack opcodes when creating a JPEG2000 file.

<b>Value</b>	<b>Meaning</b>
J2K_COLOR_SPEC_METHOD_ECS	Enumerated Color Space, The color space can be found in the ColorSpecEnumeratedColorSpace field.
J2K_COLOR_SPEC_METHOD_ICP	ICC Profile Color Space. The ICC Profile can be found in the ColorSpecICCProfileData field. If this value is set during REQ_INIT, for a pack opcode, then ColorSpecICCProfileData must be non-NULL and ColorSpecICCProfileLen should be the length, in bytes, of the profile pointed to by ColorSpecICCProfileData.

**ColorSpecPrecedence**

Reserved. Must be 0.

**ColorSpecApproximation**

Reserved. Must be 0.

**ColorSpecICCProfileLen**

The length in bytes of the ICC Profile in ColorSpecICCProfileData

**CompFileSizeHigh**

(OP\_J2KP) Upper 32 bits of the desired compressed file size in bytes; appended to the value in CompFileSize to create a 64 bit value when needed for extremely large images. Be sure to set to zero when the compressed file size fits in CompFileSize or when CompFileSize is not being specified.

---

## DIB\_OUTPUT: OP\_S2D, OP\_SE2D,

```

typedef struct {
    DWORD           Reserved0;
    BYTE PICHUGE*   AppField;
    BYTE PICHUGE*   MadeColorTable;
    BYTE PICHUGE*   PrecisionReq;
    BYTE PICHUGE*   ColorMap;
    BYTE PICHUGE*   ExifThumbnail;
    BYTE PICHUGE*   GrayMap12Bit;
    BYTE PICHUGE*   Reserved7;
    BYTE PICHUGE*   Reserved8;
    PICFLAGS        PicFlags;
    PICFLAGS        PicFlags2;
    LONG            DibSize;
    THUMBNAIL       Thumbnail;
    LONG            NumScansReq;
    LONG            NumScansDone;
    LONG            NumBytesDone;
    LONG            GraysToMake;
    LONG            PrimClrToMake;
    LONG            SecClrToMake;
    LONG            StripSize;
    LONG            WidthPad;
    LONG            LumFactor;
    LONG            ChromFactor;
    SUBSAMPLING     SubSampling;
    JPEG_TYPE       JpegType;
    LONG            AppFieldSize;
    LONG            AppFieldLen;
    LONG            NumOfPages;
    LONG            PageNum;
    LONG            Context;
    LONG            DitherType;
    LONG            YuvOutputType;
    LONG            ExifThumbnailLen;
    DWORD           ResolutionUnit;
    DWORD           XResolution;
    DWORD           YResolution;
    LONG            Brightness;
    LONG            Contrast;
    LONG            ContrastOfs;
    DWORD           EnhanceMethod;
    DWORD           Planar1Offset;
    DWORD           Planar2Offset;
} DIB_OUTPUT;

```

The **DIB\_OUTPUT** structure contains the parameters for expanding a sequential JPEG image to a DIB using the [OP\\_S2D](#) opcode.

### Fields

#### Reserved0

This field is not currently used and must be set to 0.

#### AppField

Points to a buffer to hold the application data stored with the image. This buffer is allocated and freed by the application. When expanding a JPEG image file, any APP2 data is stored into this buffer up to the size specified by **AppFieldSize**. If the buffer is too small, only **AppFieldSize** bytes are stored in the buffer. The size of the data stored is returned in the **AppFieldLen** field.

#### **MadeColorTable**

Points to a buffer that will hold the color table created for a color output image when **PF\_MakeColors** is set in **PicFlags**. This buffer holds an array of RGBQUAD structures that define the primary and secondary colors for the output image. The buffer must be large enough to hold the total number of RGBQUAD colors as specified by **PrimClrToMake** and **SecClrToMake**.

The **MadeColorTable** is only strictly necessary when both a primary and a secondary palette are desired and when the **DibSize** is 4. In that case, the secondary palette will be copied to **ColorTable** to display the DIB and the primary palette would be lost if not for **MadeColorTable** and if not copied in response to the RES\_MADE\_COLORS event.

#### **PrecisionReq**

Not used by the OP\_S2D opcode and must be set to 0.

#### **ColorMap**

Not used by the OP\_S2D opcode and must be set to 0.

#### **ExifThumbnail**

Points to EXIF Thumbnail data to be used as source for decompression when PF\_ExpandThumbnail switch is specified.

#### **GrayMap12Bit**

(OP\_SE2DPLUS) A translation table to be used to remap 12bpp grayscale data to 8bpp during decompression.

#### **Reserved7**

#### **Reserved8**

These fields are not currently used and must be set to 0.

**PicFlags**

Flags that control the Pegasus operations.

<b>Value</b>	<b>Meaning</b>
PF_AllocateComment	Causes the Pegasus routine to return with a RES_ALLOCATE_COMMENT_BUF response when a comment is encountered in the image
PF_ConvertGray	Converts a color input image to a gray scale DIB.
PF_CreateDibWithMadeColors	When <b>PF_MakeColors</b> is specified, outputs a DIB using the optimal colors.
PF_Dither	Dithers the color-mapped output
PF_EOIFound	<b>Pegasus</b> sets this flag when it encounters a JPEG EOI marker in the file
PF_MakeColors	Set to create an optimal color palette for color output image or an optimal set of gray levels for a gray scale output image (see <b>GraysToMake</b> field for additional information).
PF_NoCrossBlockSmoothing	Set if the decode operation should not smooth edges between blocks (group of pixels). An application would not normally set this flag since the decoded image looks significantly better when cross-block smoothing is applied. When this flag is not set, decoded image quality is improved by applying the proprietary techniques specified in the <b>EnhanceMethod</b> field.
PF_NoDibPad	Set if <b>Pegasus</b> is not to pad the output DIB to a DWORD boundary
PF_OnlyUseMadeColors	When <b>PF_MakeColors</b> is specified, this flag instructs <b>Pegasus</b> to output <i>only</i> the DIB created using the optimal colors. Otherwise, <b>Pegasus</b> will output a DIB while the optimal colors are being created. This first DIB output will use the colors defined by the PIC_PARM <b>ColorTable</b> if any. Otherwise it will use the colors saved with the image file, if any. Otherwise this first DIB will not be output even though <b>PF_OnlyUseMadeColors</b> is not set.
PF_SwapRB	Set to receive output data in the format R-G-B instead of B-G-R. If PF_YuvOutput is set, then the output U and V samples are swapped.
PF_UsedMMX	<b>Pegasus</b> sets this flag to indicate that Intel processor MMX optimizations were used.
PF_UsedP3	<b>Pegasus</b> sets this flag to indicate that Intel processor P3 optimizations were used.
PF_WidthPadKnown	Set if the padded length of one DIB pixel line has been set by the application. Otherwise the length is computed by <b>Pegasus</b> .
PF_YieldPut	Set if the RES_PUT_DATA_YIELD response is desired.
PF_YuvOutput	Set to receive YUV output formats. Used in conjunction with BI_UYVY, BI_YUY2, BI_YV12, or BI_IYUV

## **PicFlags2**

PF2_CompressedYUV	Use this switch for YUY2 or UYVY output if the decompressor is to “compress” the Y output intensity range to 0-235 and U/V intensity range is 0-240. Otherwise the Y/U/V output intensity ranges are 0-255 unless clipped with PF2_ClippedYUV.
PF2_ClippedYUV	Use this switch for YUY2 or UYVY output if PF2_CompressedYUV is not set and if the decompressor is to clip the Y output intensity range to 0-235 and U/V intensity range is 0-240. Otherwise the Y/U/V output intensity ranges are 0-255. This flag would be used if you knew that the original input to the compressor was compressed YUV, but that this input had been compressed as though it were full-range YUV. Setting this flag then prevents compression error from inadvertently driving decompressed intensities out of the acceptable range for compressed YUV.
PF2_MMXDisable	Set this switch to disable use of Intel processor MMX optimizations (this will also disable P3 optimizations).
PF2_P3OptDisable	Set this switch to disable use of Intel processor P3 optimizations.
PF2_ARMAsmDisable	Set this switch to disable use of ARM processor optimizations.
PF2_ARMNeonDisable	Set this switch to disable use of ARM Neon processor optimizations.
PF2_NoYccTransform PF2_RgbJpeg	<b>Pegasus</b> sets this flag (the two names have the same #defined value and therefore exactly the same behavior) during REQ_INIT if the compressed image is a 3-component, 8-bit per component image that does not have a JFIF marker, and that has an Adobe APP14 marker with a color transform value of 0, or does not have an Adobe APP14 marker but has component IDs 'R', 'G', 'B' indicating that the compressed data is in the RGB colorspace instead of in the more customary YCbCr colorspace. <b>Pegasus</b> sets this flag during REQ_INIT for a 4-component, 8-bit per component image that does not have an Adobe APP14 marker, or that has an Adobe APP14 marker with a color transform value of 0. If this flag is set during REQ_EXEC, then no color conversion will be done. The application can set or clear this flag after REQ_INIT before REQ_EXEC to override the automatic determination of colorspace and force suppression or inclusion of any color conversion.
PF2_SmoothUpsample	Set this switch to use slower but better quality smooth upsampling for chrominance. Only applies to 211 case for 8-bit output and 16-bit dithered output and to 411 case for 24-bit or 32-bit output.

PF2_RgbaOutput	Set this flag for a four-component compressed image to decompress to RGBA output (ordered B/G/R/A unless PF_SwapRB is set) instead of decompressing to CMYK output.
PF2_AcrobatCmyk	Set this flag to expand CMYK compressed images compatible with Acrobat instead of being compatible with Photoshop. Acrobat complements C/M/Y before applying the YCbCr conversion and otherwise doesn't complement. Photoshop doesn't complement C/M/Y before applying the YCbCr conversion and otherwise complements.

### DibSize

Specifies the size of the desired DIB in bits (4, 8, 16, 24 or 32). If **DibSize** is 0, then it is set by **Pegasus** to 24 if the input image is not gray scale or 8 if the input image is gray scale. If **DibSize** is 16, 24 or 32 and the input image is 8bpp gray scale or **PF\_ConvertGray** is set in **PicFlags**, then **Pegasus** sets **DibSize** to 8. If **DibSize** is 16 and input image is 12bpp gray scale, then 12 bpp gray is output within 16 bits.

### Thumbnail

Specifies the size of the thumbnail to create. This field can be one of the following values:

Value	Meaning
THUMB_NONE	Full image size
THUMB_4	1/4 image size thumbnail
THUMB_16	1/16 image size thumbnail
THUMB_64	1/64 image size thumbnail

### NumScansReq NumScansDone NumBytesDone

Not used by the OP\_S2D opcode and must be set to 0.

### GraysToMake

When **PF\_ConvertGray** is set in **PicFlags** or the input image is gray scale, then the output DIB will be gray scale. In that case, **GraysToMake** can be used to specify the number of gray levels (0 or 2..256) to create when converting the image to gray scale. A value of 0 defaults to 16 or 256 levels depending on **DibSize**.

If the output DIB will be gray scale, and if **PF\_MakeColors** is specified and **GraysToMake** is consistent with **DibSize**, then **GraysToMake** is honored by **Pegasus**.

If the output DIB will be gray scale, and if **PF\_ConvertGray** is not specified or the PIC\_PARM **Head.biClrUsed** field is 0 (no input color/gray table is supplied), and if **GraysToMake** is consistent with **DibSize**, then **GraysToMake** is honored by **Pegasus**.

### **PrimClrToMake**

If **PF\_MakeColors** is set In **PicFlags**, and **PF\_ConvertGray** is not set and the input image is a color image, **PrimClrToMake** specifies the desired number of colors to make in the primary palette [0 or 2..256]. A value of 0 defaults to 236.

### **SecClrToMake**

if **PF\_MakeColors** is set In **PicFlags**, and **PF\_ConvertGray** is not set and **PrimClrToMake** is not 0 and the input image is a color image, **SecClrToMake** specifies the desired number of colors to make in the secondary palette [0 or 2..16]. A value of 0 defaults to 16.

### **StripSize**

Specifies the minimum buffer size needed to hold one strip of output data. This field is output by **Pegasus**. Each JPEG strip is 8 or 16 image lines. The operation acts on the input data one strip at a time, so at least one complete strip of output data is to the **Put** queue at a time, except at the end of the image. Thus, this is also the minimum buffer size for the **Put** queue. If larger, and using **Q\_REVERSE**, the **Put** queue buffer size must be an integer multiple of **StripSize**.

### **WidthPad**

Specifies the width of an uncompressed DIB pixel line in bytes. **WidthPad** takes into account the number of bits needed to represent a pixel and any padding that may be required at the end of each line. **WidthPad** will be output by **Pegasus** unless the **PF\_WidthPadKnown** flag is set in **PicFlags**. If **PF\_WidthPadKnown** is set, then **WidthPad** must be input by the application. In that case, the operation will expand **Head.biWidth** bits and will then use **WidthPad** to advance to the next pixel line. This allows the application to easily expand to a rectangle within the DIB which is aligned on the left edge of a DIB. The application can expand to an arbitrary rectangle within the DIB with some additional complexity. This technique is also used to generate "raw" image data that contains specific padding or, more frequently, no padding.

### **LumFactor**

If the JPEG image were created by a PICTools operation, then the **LumFactor** specified when the file was created is returned in this field. See the **LumFactor** description in the [DIB INPUT: OP D2S](#) section of the PICTools Programmer's Reference for additional information.

### **ChromFactor**

If the JPEG images were created by a PICTools operation, then the **ChromFactor** specified when the file was created is returned in this field. See the **ChromFactor** description in the [DIB INPUT: OP D2S](#) section of the PICTools Programmer's Reference for additional information.

### **SubSampling**

Specifies sub-sampling used to create the input image.

Value	Meaning
SS_111	Cb and Cr weren't sub-sampled
SS_211	Cb and Cr were sub-sampled 2 to 1 horizontally, not vertically
SS_411	Cb and Cr were sub-sampled 2 to 1 vertically and horizontally
SS_211v	Cb and Cr were sub-sampled 2 to 1 vertically, not horizontally

If any other value is returned for **SubSampling**, the returned value was computed by **Pegasus** as:

```
( ( H * Y ) << 27 ) | ( ( V * Y ) << 22 ) |
( ( H * Cb ) << 17 ) | ( ( V * Cb ) << 12 ) |
( ( H * Cr ) << 7 ) | ( ( V * Cr ) << 2 )
```

where H and V are the sub-sampling factors defined in the JPEG specification.

### JpegType

Specifies the type of JPEG file.

Value	Meaning
JT_PIC2	PIC2 format (EIs Coded ePIC format)
JT_EXIF	EXIF format JPEG image
JT_RAW	JFIF-compliant JPEG

### AppFieldSize

Specifies the size of the buffer pointed to by the **AppField** field. If the application sets this field to -1 and the image contains APP2 marker data, the Pegasus routine will return to the application with a RES\_ALLOCATE\_APP2\_BUF response. At this time, **AppFieldLen** field will contain the length of the APP2 marker data in the image. The application can allocate a buffer, set **AppField** to point to the buffer, and continue **Pegasus** execution to retrieve all of the data associated with the APP2 marker.

### AppFieldLen

When the operation is complete, this field contains the length of APP2 marker data copied to the **AppField** buffer. When **Pegasus** returns with a RES\_ALLOCATE\_APP2\_BUF response, this field has the length of the APP2 marker data in the image. The application can allocate a buffer, set **AppField** to point to the buffer, and continue **Pegasus** execution to retrieve the data associated with the APP2 marker.

### NumOfPages

#### PageNum

#### Context

These fields are not used by the OP\_S2D opcode and must be set to 0.

### DitherType

Used if dithering. 0 = Floyd-Steinberg. (If != 1 on Solaris, it is a fast dither for colormapped output.)

### YuvOutputType

BI\_UYVY, BI\_YUY2, BI\_YV12, or BI\_IYUV. Ignored unless PF\_YuvOutput is set.

### ExifThumbnailLen

Length of the buffer pointed to by ExifThumbnail when decompressing the thumbnail from an Exif image file.

**ResolutionUnit**

ResolutionUnit is 0 if XResolution/YResolution specifies aspect ratio, otherwise 1 is inches and 2 is centimeters.

**Xresolution**

Horizontal pixels / resolution unit or numerator of aspect ratio

**Yresolution**

Vertical pixels / resolution unit or denominator of aspect ratio

**Brightness**

**Contrast**

**ContrastOfs**

Specifies the amount to increase or decrease the decompressed output image's brightness, contrast and contrast offset (as specified below) during decompression. Acceptable values are from -4095 to 4095. A value of 0 means the image is not changed for that parameter. The adjusted pixel luminance ( $L_{out}$ ) is related to the unadjusted pixel luminance ( $L_{in}$ ) as:

$$L_{out} = \text{Brightness}/16 + \tan(\pi/2 * (\text{Contrast} + 4096) / 8192) * (L_{in} - \text{ContrastOfs} / 16)$$

for 8-bit intensity samples (0-255). For 12-bit intensity samples, **Brightness** and **ContrastOfs** aren't divided by 16. The output  $L_{out}$  values are clamped to the appropriate output range. If you graph this equation with  $L_{in}$  as the horizontal axis and  $L_{out}$  as the vertical axis, changes to **Contrast** without changing **Brightness** or **ContrastOfs** have the effect of rotating the graphed line around the point (**ContrastOfs**, **Brightness**). **Brightness** raises (+) or lowers(-) the graphed line without rotating it. **ContrastOfs** shifts the graphed line right (+) or left (-) without rotating it.

**EnhanceMethod**

If **PF\_NoCrossBlockSmoothing** is not set in **PicFlags**, then this field further specifies which of the Pegasus proprietary enhancement algorithms to apply when decompressing sequential JPEG images (opcodes **OP\_S2D** and **OP\_SE2D**). Either set this field to 0 to select **Method 0** or set it to a value that is the bitwise 'OR' of the numeric values of the individual methods other than **Method 0** to select those methods; any combination of methods may be chosen. When decompressing progressive JPEG images (opcode **OP\_P2D**) this field is ignored and the default **Method 0** is used.

<b>Method</b>	<b>Meaning</b>
EM_CrossBlockSmoothing	Clean up low frequency contouring and artifacts (default)
EM_ReduceContouring	Clean up only low frequency contouring
EM_ReduceArtifacts	Clean up only low frequency artifacts
EM_ReduceBlockEdgeArtifacts	Clean up block edge artifacts

**Planar1Offset, Planar2Offset**

Set to 0 if "simple" decompression to planar YUV. Otherwise (defining "not simple") these mean the output image is re-interleaving lines or compositing the decompressed image into a larger image subrectangle. **For planar YUV the Get queue must hold the entire image.**

If Q\_Reverse is not set:

if Put.Front is offset N samples from the actual start of the entire image Y plane, set Planar1Offset to  $\langle \text{size Y plane} \rangle + N/2$  where  $\langle \text{size Y plane} \rangle$  is  $\langle \text{width} \rangle * \langle \text{height} \rangle$  for the  $\langle \text{width} \rangle$  and  $\langle \text{height} \rangle$  of the entire image. Set Planar2Offset to  $\text{Planar1Offset} + \langle \text{size Y plane} \rangle / 4$ . Thus Planar1/2Offset are the offsets from Put.Front of the U or V samples corresponding to the Y sample pointed to by Put.Front.

Else Error if Q\_Reverse is set.

---

## ZOOM\_PARMS: [OP\\_ZOOM2](#)

```
typedef struct {
    DWORD           Reserved0;
    BYTE PICHUGE*   Reserved1;
    BYTE PICHUGE*   Reserved2;
    BYTE PICHUGE*   Reserved3;
    BYTE PICHUGE*   Reserved4;
    BYTE PICHUGE*   Reserved5;
    BYTE PICHUGE*   Reserved6;
    BYTE PICHUGE*   Reserved7;
    BYTE PICHUGE*   Reserved8;
    PICFLAGS        PicFlags;
    PICFLAGS        PicFlags2;
    LONG            NewWidth;
    LONG            NewHeight;
    WORD            Mode;
    WORD            NewBitCount;
    void PICFAR*    WorkArea;
    BITMAPINFOHEADER BiOut;
    DWORD           Reserved;
    REGION          RegionOut;
} ZOOM_PARMS;
```

The **ZOOM\_PARMS** structure contains the parameters for [OP\\_ZOOM2](#) opcode used for changing the dimensions of a DIB, converting a DIB to gray scale or converting a DIB to halftone. The ZOOM\_PARMS structure is used as follows.

### Fields

#### Reserved0

This field is not currently used and must be set to 0.

#### Reserved1

#### Reserved2

#### Reserved3

#### Reserved4

#### Reserved5

#### Reserved6

#### Reserved7

#### Reserved8

These fields are not currently used and must be set to 0.

**PicFlags**

Flags that control the zoom operation.

Value	Meaning
PF_FastZoom	Use fast zoom mode.
PF_ConvertGray	Convert image to grayscale.
PF_PreserveBlack	In fast zoom mode with 1-bit input, ensures preservation of all black pixel values. In shrinking a 1-bit input image to a grayscale output image, ensures that black pixel values are not overly diluted.
PF_YieldGet	Set if the RES_GET_DATA_YIELD response is desired.

**PicFlags2**

This field is not currently used and must be set to 0.

**NewWidth**

This field can be used to specify the new image width, although use of the **Width** field of the output region described by **RegionOut** field is preferred (**REGION** or **REGION2**—see the comments on **REGION2** below). If **NewWidth** is nonzero and the **Width** field of the output region is zero, then the **Width** field of the output region is reset to the value of **NewWidth**. If both **NewWidth** and the **Width** field of the output region are nonzero, then they must be equal.

**NewHeight**

This field can be used to specify the new image height, although use of the **Height** field of the output region described by **RegionOut** field is preferred (**REGION** or **REGION2**—see the comments on **REGION2** below). If **NewHeight** is nonzero and the **Height** of the output region is zero, then the **Height** field of the output region is reset to the value of  $\text{abs}(\text{NewHeight})$  and the **RF\_TopDown** bit of **RegionOut.Flags** is set if **NewHeight** is negative. If both **NewHeight** and the **Height** field of the output region are nonzero, then the **Height** field of the output region must equal  $\text{abs}(\text{NewHeight})$  and the **RF\_TopDown** bit of **RegionOut.Flags** must be set if and only if **NewHeight** is negative.

**Mode**

If **Mode** is 0, then the new image has dimensions *new\_width* (as given by **NewWidth** or the **Width** field of the output region) columns by *new\_height* (as given by  $\text{abs}(\text{NewHeight})$  or the **Height** field of the output region) lines.

If **Mode** is nonzero, then the new image has dimensions

$$\text{Input\_width} * ( \text{new\_width} / \text{new\_height} )$$

columns (rounded) by

$$\text{input\_height} * ( \text{new\_width} / \text{new\_height} )$$

rows (rounded). The REQ\_INIT operation returns the new image dimensions in **NewWidth** and **NewHeight**, and in **BiOut.biWidth** and **BiOut.biHeight**, as well as in the **Width** and **Height** fields of the output region.

Both *new\_width* and *new\_height* must be greater than 0 and less than 65536.

### **NewBitCount**

This field can be used to specify the new image bit-depth, although use of the **Bpp** field of the output region is preferred (**REGION** or **REGION2**—see the comments on **REGION2** below). If **NewBitCount** is nonzero and the **Bpp** field of the output region is zero, then the **Bpp** field of the output region is reset to the value of **NewBitCount**. If both **NewBitCount** and the **Bpp** field of the output region are nonzero, then they must be equal.

### **WorkArea**

This is reserved.

### **BiOut**

Description of the output image. This component is set by the initial REQ\_INIT request. It is recommended that the calling program refer to the output region (**REGION** or **REGION2**—see the comments on **REGION2** below) rather than **BiOut** although both are set. **BiOut** contains the BITMAPINFOHEADER for the new image. In cases where the new image has a color palette, the palette is stored in the ColorTable element of the PIC\_PARM structure. This field is unused for gray-level images using two bytes per pixel. In other cases information is redundant with that provided in RegionOut.

### **Reserved**

This field is not currently used and must be set to 0.

### **RegionOut**

Description of the output image. This component is set by the initial REQ\_INIT request. It is recommended that the calling program refer to **RegionOut** rather than **BiOut** although both are set. In cases where the new image has a color palette, the palette is stored in the ColorTable element of the PIC\_PARM structure.

The following components of **RegionOut** may be set before the REQ\_INIT operation to specify properties of the output image:

**Width**

**Height**

**Bpp**

**PixType**

**Flags (RF\_TopDown, RF\_2Byte, RF\_Packed).**

See [REGION](#) for explanation of values of **PixType** and **Flags** and their meanings. Values used for OP\_ZOOM2 are: PT\_CM, PT\_GRAY, PT\_GRAYM, PT\_RGB, PT\_RGBA, PT\_RGB555, and PT\_RGB565. See *Default Values* below.

The **Stride** element of **RegionOut** may be set by the user between the REQ\_INIT and REQ\_EXEC operations.

The **REGION2** structure may be used in place of the **REGION** for specifying both the input and output rectangles. To do this, recast the addresses of both the **RegionIn** element of the **PIC\_PARM** structure and the **RegionOut** element of the **ZOOM\_PARMS** structure as type **REGION2\*** and set the appropriate fields of the corresponding **REGION2**. The **ParmVerMinor** field of the **PIC\_PARM** structure should be set to a value not less than 4.

#### *Grayscale Conversion*

The image is converted to gray-level if the **PF\_ConvertGray** flag is set, or if the output pixel type is **PT\_GRAY** or **PT\_GRAYM**. Gray-level conversion is also forced:

- When the output image is colormapped (pixel type **PT\_CM**) and the input image is not;
- When the output image is colormapped with fewer bits per pixel than the input image;
- When the input image is 1 bit per pixel and the output image is larger in size than the input.

#### *Halftoning*

The image is halftoned (depicted with black and white dots of various sizes) if the input bit-depth is greater than 1 and the output bit-depth is 1. See also [OP\\_BINARIZE](#) for more flexible halftoning operations.

#### *Signed Pixel Values*

Signed pixel values may be used for either input or output with the **PT\_GRAY** and **PT\_GRAYM** pixel types.

#### *YUYV*

DIBs in YUYV format can be increased or decreased in size; however when the input is YUYV, the output must be YUYV, and vice-versa. See also the note below on fast mode. With YUYV, both input and output widths must be even. In Mode 0, an odd output width results in an error condition. In Mode 1, the output width is rounded to the nearest even value. In addition, when zooming a cropped YUYV image, **CropXoff** is rounded down to the nearest even value.

#### *Fast mode*

The image is resized using fast mode (resampling) if the **PF\_FastZoom** flag is set or in the following situations:

- The input and output bit-depths are both 1;
- The input bit depth is 1 and the output width or height is larger than the original;
- Input and output images are both colormapped, and the output image has no fewer bits per pixel than the input image;
- Input and output images are both gray-level with more than eight bits of precision per sample.
- Input and output images are YUYV and either the height or width is increased.

### *Cropping*

Two modes of cropping are available, triggered by the **F\_InputCrop** (or **F\_Crop**) and **F\_OutputCrop** flags of the **Flags** field of the **PIC\_PARM** structure.

If **F\_InputCrop** is set, then the image is cropped *before* zooming. The cropping window is specified by the **IOCropXoff**, **IOCropYoff**, **IOCropWidth**, and **IOCropHeight** parameters of the **PIC\_PARM** structure. The result of this operation is identical to separately cropping the input image, followed by a zoom operation. Values of **IOCropXoff** or **IOCropYoff** exceeding the image dimensions result in an error. Zero values of **IOCropWidth** or **IOCropHeight** result in an error. Values of **IOCropWidth** or **IOCropHeight** exceeding the image dimensions are truncated to the image dimensions.

Alternatively, **F\_Crop** may be set and the cropping window is specified by the **CropXoff**, **CropYoff**, **CropWidth**, and **CropHeight** parameters of the **PIC\_PARM** structure. This yields results identical to the use of **F\_InputCrop**, provided the 16-bit depth of **CropXoff**, etc. is not exceeded. However, for **OP\_ZOOM2** and other opcodes that support it (see opcode list in the **F\_InputCrop** description, in the **PIC\_PARM** section), the use of **F\_InputCrop** is recommended over **F\_Crop**. Values of **CropXoff** or **CropYoff** exceeding the image dimensions result in an error. Zero values of **CropWidth** or **CropHeight** result in an error. Values of **CropWidth** or **CropHeight** exceeding the image dimensions are truncated to the image dimensions.

If **F\_OutputCrop** is set, then the image is cropped after zooming. The cropping window is specified by the **IOCropXoff**, **IOCropYoff**, **IOCropWidth**, and **IOCropHeight** parameters of the **PIC\_PARM** structure. Note that these are specified relative to the zoomed image coordinates. The result of this operation is identical to zooming the image, followed by a separate cropping operation.

Output cropping is recommended for tiling the zoomed image.

For output cropping, values of **IOCropXoff** or **IOCropYoff** exceeding the image dimensions result in an error. Zero values of **IOCropWidth** or **IOCropHeight** result in an error. Values of **IOCropWidth** or **IOCropHeight** exceeding the image dimensions are truncated to the image dimensions.

At most one of **F\_InputCrop**, **F\_OutputCrop**, or **F\_Crop** may be set in any single zoom operation.

The input region and output region dimensions are calculated according to the specified cropping parameters. The **CropOutputXoff**, **CropOutputYoff**, **CropOutputWidth**, and **CropOutputHeight** parameters may be set between the **REQ\_INIT** and **REQ\_EXEC** phases. Note that the overall image dimensions are calculated in the **REQ\_INIT** phase.

For a YUYV image, **CropXoff**, **CropOutputXoff**, **CropWidth**, and **CropOutputWidth** must be even-valued.

### *Default values*

The **Bpp** field of the output region, the **PixType** field of the output region, the output region width, and/or the output region height take default values if set to zero before the **REQ\_INIT** operation. These default values are determined as follows:

If the **PixelFormat** field of the output region is set to zero, its value is determined from the value of the **Bpp** field of the output region and the **PF\_ConvertGray** value thus:

- **PT\_CM** if  $1 < \text{output Bpp} \leq 8$  and **PF\_ConvertGray** is not set;
- **PT\_GRAY** if **PF\_ConvertGray** is set or  $8 < \text{output Bpp} \leq 16$ ;
- **PT\_RGB** if **output Bpp**=24;
- **PT\_RGBA** otherwise.

If the **Bpp** field of the output region is set to zero, its value is determined from the value of the **PixelFormat** field of the output region thus:

- 8 if the output **PixelFormat**=**PT\_CM** or the output **PixelFormat**=**PT\_GRAY**;
- 15 if the output **PixelFormat**=**PT\_RGB555**;
- 16 if the output **PixelFormat**=**PT\_RGB565** or the output **PixelFormat**=**PT\_GRAYM**;
- 24 if the output **PixelFormat**=**PT\_RGB**;
- 32 if the output **PixelFormat**=**PT\_RGBA**.

If the output **Bpp** and the output **PixelFormat** are both set to zero, their values are determined thus:

- The output **Bpp**=1 and the output **PixelFormat**=**PT\_CM** if the input image is 1 bit per pixel;
- The output **Bpp**=8 and the output **PixelFormat**=**PT\_GRAY** if **PF\_ConvertGray** is set;
- The output **Bpp**=16 and the output **PixelFormat**=**PT\_YUYV** if the input **PixelFormat** equals **PT\_YUYV**;
- The output **Bpp**=24 and the input **PixelFormat**=**PT\_RGB** otherwise.

If the output width or height are set to zero, their values are set to twice the source image width or height, respectively.

Restrictions on both input and output images (after setting default values)

- **PixelFormat** must be one of **PT\_CM**, **PT\_GRAY**, **PT\_GRAYM**, **PT\_RGB**, **PT\_RGBA**, **PT\_YUYV**, **PT\_RGB555**, or **PT\_RGB565**;
- **Bpp** may not take values from 17 to 23, from 25 to 31, or exceeding 32;
- **Bpp** must equal 24 when **PixelFormat**=**PT\_RGB** and vice-versa;
- **Bpp** must equal 32 when **PixelFormat**=**PT\_RGBA** and vice-versa;
- **Bpp** must equal 15 or 16 when **PixelFormat**=**PT\_RGB555**;
- **Bpp** must equal 16 when **PixelFormat**=**PT\_RGB565** or **PixelFormat**=**PT\_YUYV**;
- **Bpp** may not exceed 8 when **PixelFormat**=**PT\_CM**;
- **Bpp** may not exceed 16 when **PixelFormat**=**PT\_GRAY**;
- **PixelFormat** may equal **PT\_GRAYM** only if  $8 < \text{Bpp} \leq 16$ , or if  $1 < \text{Bpp} \leq 8$  and **PF\_2Byte** is set;

- Packed pixels (**RF\_Packed**) may be used on both input and output images only for bit-depths of 1 or 4.
- Signed pixels (**RF\_Signed**) may be used on both input and output images only pixel types PT\_GRAY or PT\_GRAYM.

Other restrictions:

- The output **PixType** may equal PT\_RGBA only if the input image also has pixel type RGBA;
- The output **PixType** may equal PT\_YUYV only if the input image also has pixel type PT\_YUYV, and vice-versa;
- Minimum input image dimensions are 4×4;
- One-bit-per-pixel images must have packed pixels.
- **RegionOut.Height** and **RegionOut.Width** must be greater than zero for **Mode**=0 or 1.
- For compatibility with earlier versions the **RF\_Packed** flag of **RegionOut** is set when **NewBitCount** is nonzero and the output **Bpp** is zero, or when both **NewBitCount** and the output **Bpp** are zero and the input image has bit-depth 1 (forcing the output image also to have bit-depth 1).

---

## PEGQUERY: PegasusQuery

```

typedef struct {
    DWORD           Reserved0;
    BYTE PICHUGE*   Reserved1;
    BYTE PICHUGE*   Reserved2;
    BYTE PICHUGE*   Reserved3;
    BYTE PICHUGE*   Reserved4;
    BYTE PICHUGE*   Reserved5;
    BYTE PICHUGE*   Reserved6;
    BYTE PICHUGE*   Reserved7;
    BYTE PICHUGE*   Reserved8;
    PICFLAGS        PicFlags;
    PICFLAGS        PicFlags2;
    LONG            NumOfPages;
    LONG            PageNum;
    DWORD           BitFlagsReq;
    DWORD           BitFlagsAck;
    DWORD           ImageSize;
    DWORD           AuxSize;
    DWORD           ImageNum;
    DWORD           NumImages;
    DWORD           SOIMarker;
    SUBSAMPLING     SubSampling;
    REGION          Region;
    BYTE            LumFactor;
    BYTE            ChromFactor;
    BYTE            AllowedBitErr;
    BYTE            TiffFirstByte;
    WORD            ClusterHeaderVersion;
    WORD            DjVuImageType;
    DWORD           TIFFCompression;
    DWORD           TIFFPhotometricInterpretation;
    DWORD           TIFFIFDOffset;
    WORD            PDFMajorVersion;
    WORD            PDFMinorVersion;
    DWORD           PixelDataOffset;
    DWORD           PixelDataSize;
    BYTE            TIFFOrientation;
    BYTE            TIFFPredictor;
    BYTE            TIFFFillOrder;
    BYTE            TIFFT4Options;
    WORD            NumPartitions;
} PEGQUERY;

```

The **PEGQUERY** structure supplies parameters for the **PegasusQuery** function used to retrieve information about an image.

### Fields

#### **Reserved0**

This field is not currently used and must be set to 0.

**Reserved1**  
**Reserved2**  
**Reserved3**  
**Reserved4**  
**Reserved5**  
**Reserved6**  
**Reserved7**  
**Reserved8**

These fields are not currently used and must be set to NULL.

#### PicFlags

Value	Meaning
PF_IsProtected	Set by PegasusQuery if the input PIC2-format file requires a password. OP_D2W, OP_LIP3 (unless METHOD_JPEG and PF_JpegOnly are set), and OP_D2SE/OP_D2SEPLUS (if JpegType = JT_PIC2) create PIC2 files.
PF_IsTransparency	Set by PegasusQuery if the input GIF file has a transparent color.
PF_IsGray	Set by PegasusQuery if the input PNG image is grayscale.

#### PicFlags2

Value	Meaning
PF2_IsDCX	Set by PegasusQuery if the input image is a DCX-format file. Head.biCompression is set according to the PCX image in the file whose settings are being queried according to ImageNumber..
PF2_IsExif	Set by PegasusQuery if the input JPEG file is an EXIF file.
PF2_SpecialTIFF	Input to PegasusQuery if the input TIFF file requires special handling as a result of non-compliance with the TIFF specification.
PF2_3D_Slices	Set by PegasusQuery if the input JPEG 2000 file is comprised of 3D slices
PF2_SkipJP2Header	Set by PegasusQuery if the input JPEG2000 file contains only a JPEG2000 codestream (with no JP2/JPX header boxes).

#### NumOfPages

Not used.

#### PageNum

Not used.

#### BitFlagsReq

Specifies which image information the application desires **PegasusQuery** to return. **PegasusQuery** returns TRUE if it is able to return all the information requested by **BitFlagsReq**, otherwise **PegasusQuery** returns FALSE. See **BitFlagsAck** for a description of the acceptable flags for this field.

#### BitFlagsAck

Specifies which image information was returned by **PegasusQuery**. If a flag is not set in **BitFlagsAck**, then the corresponding data should not be used. For example, if **QBIT\_BICLRUSED** is not set, then the data in the **Head.biClrUsed** field in the **PIC\_PARM** structure is not valid.

Flag	Corresponding PicParm Field	Meaning
QBIT_AUXSIZE	u.QRY.AuxSize	Size of the auxiliary information in the image such as comments
QBIT_BIBITCOUNT	Head.biBitCount	BITMAPINFOHEADER
QBIT_BICOMPRESSION	Head.biCompression	BITMAPINFOHEADER
QBIT_BICLRIMPORTANT	Head.biClrImportant	BITMAPINFOHEADER
QBIT_BICLRUSED	Head.biClrUsed	BITMAPINFOHEADER
QBIT_BIHEIGHT	Head.biHeight	BITMAPINFOHEADER
QBIT_IMAGESIZE	u.QRY.ImageSize	Size of the information that must be processed to display the image.
QBIT_BIPLANES	Head.biPlanes	BITMAPINFOHEADER
QBIT_BISIZE	Head.biSize	BITMAPINFOHEADER
QBIT_BISIZEIMAGE	Head.biSizeImage	BITMAPINFOHEADER
QBIT_BIWIDTH	Head.biWidth	BITMAPINFOHEADER
QBIT_BIXPIXELSPERMETER	Head.biXPelsPerMeter	BITMAPINFOHEADER
QBIT_BIYPIXELSPERMETER	Head.biYPelsPerMeter	BITMAPINFOHEADER
QBIT_COMMENT	Comment	Comment associated with the image
QBIT_NUMIMAGES	u.QRY.NumImages	Not used.
QBIT_PALETTE	ColorTable	The color table stored with the image.
QBIT_SOIMARKER	u.QRY.SOIMarker	File offset in a TIFF JPEG file of the JPEG SOI marker.

### ImageSize

Ordinarily, the size of the image file. This is the number of bytes required starting from the beginning of the file in order to access all image header data and all image data. **ImageSize** may not include information appended to the end of the file, in some formats, whose length is not reflected in the image header data. For TIFF files, **ImageSize** may include data from more than one image, depending upon the organization of image data in the TIFF file.

### AuxSize

Size of the buffer needed to hold all auxiliary data.

### ImageNum

Not used.

### NumImages

This field is currently always 1 except for TIFF files containing multiple images. In that case, this field is set to the number of images in the TIFF file whenever the **ImageNumber** field in **PIC\_PARM** is set to 0.

### SOIMarker

The file offset of the JPEG SOI marker in a TIFF JPEG file.

**SubSampling**

If present and lossy JPEG

**Region**

**LumFactor**

If present and lossy JPEG

**ChromFactor**

If present and lossy JPEG

**AllowedBitErr**

If lossless JPEG

**TIFFFirstByte**

Set to first byte of TIFF file ('I' or 'M') if TIFFIFDOffset is being used

**ClusterHeaderVersion**

If present for PIC2 files – primarily BI\_WAVE files produced by OP\_W2D

**DjVuImageType**

Not used

**TIFFCompression**

If TIFF

**TIFFPhotometricInterpretation**

If TIFF

**TIFFIFDOffset**

Set to the offset of first byte of Get queue within TIFF file if the first byte in the Get queue is the start of a TIFF IFD in the TIFF file. Then TIFFFirstByte must be 'I' or 'M' – the value of the first byte of the TIFF file.

**PDFMajorVersion**

If PDF

**PDFMinorVersion**

If PDF

**PixelDataOffset**

If TIFF and there is only one strip, then this is set to the offset within the TIFF file of that strip.

**PixelDataSize**

If TIFF and there is only one strip, then this is set to the size in bytes of the strip.

**TIFFOrientation**

If TIFF

**TIFFPredictor**

If TIFF

**TIFFFillOrder**

If TIFF

**TIFFT4Options**

If TIFF

**NumPartitions**

For a JPEG2000 image; The number of partitions that is required to extract the components of the image. It is set appropriately depending upon whether there are multiple gray components, or RGB components (which are combined into single partitions), or 3D slices (where PF2\_3D\_Slices is set in PicFlags2)

---

## REGION

```
typedef struct {
    BYTE          Sig;
    BYTE          Interlace;
    BYTE          BitErr;
    REGION_FLAGS Flags;
    PIXEL_TYPE    PixType;
    BYTE          Bpp;
    WORD          Width;
    WORD          Height;
    SHORT        Stride;
    DWORD        Offset;
} REGION;
```

The **REGION** structure defines the contents of an image partition. This structure is used to reference general-purpose image regions.

### Fields

#### Sig

Must be 0.

#### Interlace

0 if not interlaced, 1 for PNG interface pattern, 2 for GIF interlace pattern. For PT\_GRAY/PT\_GRAYM JPEG 2000 images, bits 2-4 can hold the vertical subsampling factor minus one and bits 5-7 plus can hold the horizontal subsampling factor minus one.

#### Flags

Attributes of the region.

Value	Meaning
RF_TopDown	Set if image stored top line first, bottom last
RF_NonInter	(unused for now) Set if image is non-interleaved
RF_MakeGray	Set if region to be treated as gray scale
RF_SwapRB	Set to treat input component samples as interleaved in RGB order instead of interleaved in BGR order (Windows DIBs are in BGR order)
RF_2Byte	Set for <= 8 Bpp images if each sample occupies two bytes
RF_Signed	Set if sample values are to be interpreted as signed values
RF_BigEndian	Set when sample values occupy two bytes and the bytes are arranged in big-endian (Motorola) order instead of little-endian (Intel) order
RF_CM2RGB	Set if a color-mapped image is to be converted to a 3-component RGB image
RF_Channel	Set if a single color channel is to be used
RF_ChLo	If RF_Channel is set, ChHi is two bits whose value 0..3 indicate which color channel
RF_ChHi	If RF_Channel is set, ChLo is two bits whose value 0..3 indicate which color channel
RF_Packed	Not implemented

**PixType**

Type of pixels which comprise this region

<b>Value</b>	<b>Meaning</b>
PT_NONE	Undefined or compressed pixels
PT_CM	Color mapped up to 256 colors
PT_GRAY	Gray scale up to 16 bits
PT_GRAYM	Gray scale up to 16 bits (big-endian)
PT_RGB	RGB 24, 48 bit (blue is low, red is high, Intel-ordered)
PT_RGBM	RGB 48 bit (blue is low, Motorola-ordered)
PT_RGB555	RGB 16 bit (xxxxrrggggbbbbb – blue is low bits)
PT_RGB565	RGB 16 bit (rrrrrgggggbbbbb – blue is low bits)
PT_CMYK	CMYK 32 bit (cyan is low, black is high)
PT_GRAYA	Gray scale 1-16 bits with alpha, Intel-ordered
PT_GRAYAM	Gray scale 9-16 bits with alpha, Motorola-ordered
PT_RGBA	RGBA 32, 64 bit (blue is low, alpha is high, Intel-ordered)
PT_RGBAM	RGBA 64 bit (blue is low, alpha is high, Motorola-ordered)

**Bpp**

Meaningful bits per pixel

**Width**

Width of region in pixels > 0

**Height**

Height of region in pixels > 0

**Stride**

Width in bytes of area containing the region. This value is at least  $(\text{Width} * \text{Bpp} + 7) / 8$ . The stride may be larger than these values if the region is contained within a larger region or if each line is padded (e.g. to a DWORD boundary for Windows bitmaps.).

**Offset**

Offset in bytes of start of the region line whose offset has the smallest value. This is the offset of the top image line for a non-interlaced RF\_TopDown image. It is the offset of the bottom image line for a non-interlaced image which is not RF\_TopDown.

---

## REGION2

```
typedef struct {
    BYTE          Sig;
    REGION_FLAGS  Flags;
    PIXEL_TYPE    PixType;
    BYTE          Bpp;
    DWORD         Width;
    DWORD         Height;
    DWORD         Stride;
} REGION2;
```

The **REGION2** structure defines the contents of an image partition. This structure is used to reference general-purpose image regions. The structure is a replacement for the **REGION** structure to allow for Width/Height/Stride larger than 65535. Read the opcode description to determine if an opcode supports **REGION2**. **If supported, and the opcode supports both REGION2 and REGION, set ParmVerMinor to at least 4 to use REGION2 instead of REGION.** Then use the accessor macros Region2() and Region2Ptr() defined in PIC2.H as, for example:

```
Region2(pp.u.Region).Stride =
    Region2(pp.u.Region).Width *
    ( ( Region2(pp.u.Region).Bpp + 7 ) / 8 );
```

or

```
Region2Ptr(&pp.u.Region)->Stride =
    Region2Ptr(&pp.u.Region)->Width *
    ( ( Region2Ptr(&pp.u.Region)->Bpp + 7 ) / 8 );
```

### Fields

#### Sig

Must be 2.

#### Flags

Attributes of the region.

Value	Meaning
RF_TopDown	Set if image stored top line first, bottom last
RF_NonInter	(unused for now) Set if image is non-interleaved
RF_MakeGray	Set if region to be treated as gray scale
RF_SwapRB	Set to treat input component samples as interleaved in RGB order instead of interleaved in BGR order (Windows DIBs are in BGR order)
RF_WhitelsZero	Set for a bilevel (1bpp) image to indicate that 0 samples are white instead of the default of 0 samples being black.
RF_2Byte	Set for <= 8 Bpp images if each sample occupies two bytes
RF_Signed	Set if sample values are to be interpreted as signed values

RF_BigEndian	Set when sample values occupy two bytes and the bytes are arranged in big-endian (Motorola) order instead of little-endian (Intel) order
RF_CM2RGB	Set if a color-mapped image is to be converted to a 3-component RGB image
RF_Channel	Set if a single color channel is to be used
RF_ChLo	If RF_Channel is set, ChHi is two bits whose value 0..3 indicate which color channel
RF_ChHi	If RF_Channel is set, ChLo is two bits whose value 0..3 indicate which color channel
RF_Packed	Not implemented

### PixType

Type of pixels which comprise this region

Value	Meaning
PT_NONE	Undefined or compressed pixels
PT_CM	Color mapped up to 256 colors
PT_GRAY	Gray scale up to 16 bits
PT_GRAYM	Gray scale up to 16 bits (big-endian)
PT_RGB	RGB 24, 48 bit (blue is low, red is high, Intel-ordered)
PT_RGBM	RGB 48 bit (blue is low, Motorola-ordered)
PT_RGB555	RGB 16 bit (xrrrrgggggbbbb – blue is low bits)
PT_RGB565	RGB 16 bit (rrrrrgggggbbbb – blue is low bits)
PT_CMYK	CMYK 32 bit (cyan is low, black is high)
PT_GRAYA	Gray scale 1-16 bits with alpha, Intel-ordered
PT_GRAYAM	Gray scale 9-16 bits with alpha, Motorola-ordered
PT_RGBA	RGBA 32, 64 bit (blue is low, alpha is high, Intel-ordered)
PT_RGBAM	RGBA 64 bit (blue is low, alpha is high, Motorola-ordered)
PT_RGB101010	RGB 32-bit (xxrrrrrrrrgggggggggbbbbbbbbb) – blue is low bits)
PT_RGBE	The Radiance .HDR format floating point with an R, then G, then B mantissa followed by a shared exponent.
PT_CMYKA	PT_CMYK followed by an alpha channel
PT_GRAY_MSFIXED	Gray scale in 16-bit or 32-bit Microsoft fixed-point format
PT_GRAY_FLOAT	Gray scale in 16-bit (HALF) or 32-bit (single-precision IEEE) floating point format
PT_RGB_MSFIXED	PT_RGB in 16-bit or 32-bit Microsoft fixed-point format
PT_RGB_FLOAT	PT_RGB in 16-bit (HALF) or 32-bit (single-precision IEEE) floating point format
PT_RGBA_MSFIXED	PT_RGBA in 16-bit or 32-bit Microsoft fixed-point format
PT_RGBA_FLOAT	PT_RGBA in 16-bit (HALF) or 32-bit (single-precision IEEE) floating point format

### Bpp

Meaningful bits per pixel

**Width**

Width of region in pixels

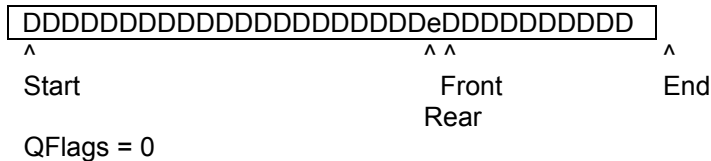
**Height**

Height of region in pixels

**Stride**

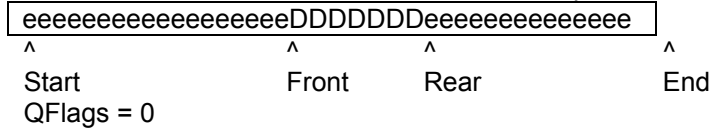
Width in bytes of area containing the region. This value is at least  $( \text{Width} * \text{Bpp} + 7 ) / 8$ . The stride may be larger than these values if the region is contained within a larger region or if each line is padded (e.g. to a DWORD boundary for Windows bitmaps)).





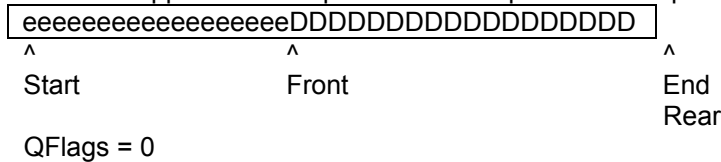
**More data is taken from the queue.**

The data consumer takes more data out of the queue.

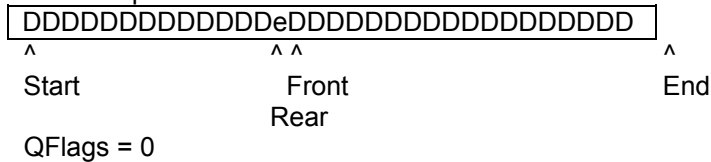


**More data is again added to the queue.**

The data supplier fills the queue in two steps. First the queue is filled from Rear to End.



Then the queue is filled from Start.



**Circular Queue Code Example**

Following is a circular get queue example showing the Forward case. The Get Queue initialization code looks like this:

```

// Get queue initialization
// allow for one byte more so Front == Rear means empty
pbQBuffer = malloc(dwQDataDesired + 1);
PicParm.Get.Start = pbQBuffer;
PicParm.Get.End   = pbQBuffer + dwQDataDesired + 1;
PicParm.Get.Front = PicParm.Get.Start;
PicParm.Get.Rear  = PicParm.Get.Front;
  
```

The application code that will be invoked when the opcode requires more data to be added to its Get Queue is presented below. In this code, we assume that the data being copied to the Get Queue exists in a buffer tracked by these two global variables:

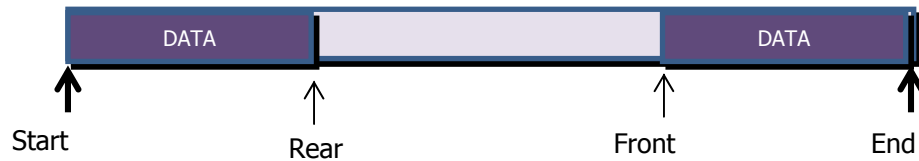
```

BYTE *InputData      : beginning of the available data
DWORD InputDataLength: current size of available data
  
```

Every time the code is invoked, InputData will grow until it reaches the end of available data. Conversely InputDataLength will decrease until it reaches zero meaning all data has been consumed.

We can assume that this code will not be called if the Get Queue is full; therefore, there are 2 scenarios that the code needs to deal with:

- 1) The buffer is empty. We know by checking if `Rear==Front`. In this case, just fill the buffer
- 2) The buffer is partially full. In this case there are two possible situations. We determine which one it is by looking at the relative position of `Front` and `Rear`:
  - a. `Rear` is before `Front`: the empty space (to be filled with data) is contiguous. In this case, we need to fill the empty space from `Rear` to `Front-1`.



- b. `Front` is before `Rear`: the data in the buffer is contiguous. In this case, we need to fill the empty spaces: `Rear` to `End` and `Start` to `Front-1`



The code to add data to the Get Queue looks like this:

```

if (PicParm.Get.Front == PicParm.Get.Rear) {
    // this is the empty queue scenario...
    PicParm.Get.Front = PicParm.Get.Rear = PicParm.Get.Start;
    CopyLen = PicParm.Get.End - PicParm.Get.Start - 1;
    CopyToRear_And_AdjustInput(PicParm, CopyLen);
} else { // partially full scenario
    if (PicParm.Get.Rear < PicParm.Get.Front) {
        // empty space is contiguous: rear->front-1
        CopyLen = PicParm.Get.Front - 1 - PicParm.Get.Rear;
        CopyToRear_And_AdjustInput(PicParm, CopyLen);
    } else {
        // empty space has 2 segments... rear->end
        CopyLen = PicParm.Get.End - PicParm.Get.Rear;
        CopyToRear_And_AdjustInput(PicParm, CopyLen);
        // start->front-1
        PicParm.Get.Rear = PicParm.Get.Start;
        CopyLen = PicParm.Get.Front - 1 - PicParm.Get.Rear;
        CopyToRear_And_AdjustInput(PicParm, CopyLen);
    }
}

```

The utility function that copies from the input buffer into the queue and adjusts InputData and InputDataLength as needed, is shown below:

```
void CopyToRear_And_AdjustInput(PIC_PARM PicParm, DWORD CopyLen)
{
    if ( CopyLen > InputDataLength )
        CopyLen = InputDataLength;
    memcpy(PicParm.Get.Rear, InputData, CopyLen);
    InputData += CopyLen;
    InputDataLength -= CopyLen;
    PicParm.Get.Rear += CopyLen;
}
```

## APPENDIX 2. Handling Pegasus() Responses

In some scenarios, Pegasus may send an intermediate response to the application before the requested operation is completed and the final response (RES\_DONE or RES\_ERROR) is sent.

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

### **Callback function (DeferFn) mode.**

- 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 back to the application.
- 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
- This mode is available in all platforms and CPU architectures.
- This mode is selected by setting the `F_UseDeferFn` flag in `PIC_PARM.Flags` and assigning a value to `PIC_PARM.DeferFn`.

### **Coroutine mode.**

- • In this mode, the application's call to Pegasus returns whenever an intermediate response is to be sent back to the application. If the return code is different from RES\_DONE or RES\_ERROR, then it is an intermediate response.
- • 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.
- • This is the mode whenever the `F_UseDeferFn` flag is not set in `PIC_PARM.Flags`. On iOS and Android this mode is based on a stack switching scheme where the opcode uses an internally-allocated stack that is distinct from the application stack. This is known as Stack Based Coroutine Mode.