

# Intel<sup>®</sup> JPEG Library

Using the IJL with JPEG  
Compressed FlashPix Files



Version 1.4  
June22,1999

Copyright © 1998 , 1999Intel Corporation. All rights reserved.  
Intel Corporation, 5200 N.E. Elam Young Parkway, Hillsboro, OR 97124-6497

Intel Corporation assumes no responsibility for errors or omissions in this guide. Nor does Intel make any commitment to update the information contained herein.

\* Other product and corporate names may be trademarks of other companies and are used only for explanation and the owners' benefit, without intent to infringe.

# Table of Contents

<b>1. Introduction .....</b>	<b>1</b>
1.1 About This Document .....	1
1.2 Nature of Product .....	1
1.3 Technical Support and Feedback .....	1
<b>2. FlashPix Overview .....</b>	<b>2</b>
2.1 FlashPix Storage .....	2
<b>3. Decoding JPEG FlashPix Tiles .....</b>	<b>3</b>
3.1 Example .....	3
<b>4. Authoring JPEG FlashPix Tiles .....</b>	<b>11</b>
4.1 Example .....	11

# 1. Introduction

## 1.1 About This Document

This document explains how to optimally use of the Intel® JPEG Library (IJL) to encode and decode JPEG compressed FlashPix\* files.

It is assumed that the reader has a working knowledge of the software development process and the C/C++ programming language. The reader should be familiar with the FlashPix file format specification and have experience with compound document access. Some familiarity with digital imaging, software development for the Microsoft\* Windows\* 95 operating system, and the Microsoft Foundation Classes\* application framework may also be useful.

## 1.2 Nature of Product

The IJL is a software library for application developers that provides high performance JPEG encoding and decoding of full color, and grayscale, continuous-tone still images.

The IJL was designed for use on Intel® architecture platforms and has been tuned for high performance and efficient memory usage. Additionally, the IJL was developed to take advantage of MMX™ technology if present.

The IJL provides an easy-to-use programming interface without sacrificing low-level JPEG control to advanced developers. The IJL also includes a substantial amount of functionality that is not included in the ISO JPEG standard. This added functionality is typically necessary when working with JPEG images, and includes pre-processing and post-processing options like sampling and color space conversions.

## 1.3 Technical Support and Feedback

Your feedback on the IJL is very important to us. We will strive to provide you with answers or solutions to any problems you might encounter. To give your feedback, or to report any problems with installation or use, please contact one of the following:

- Support Hotlines:  
North American Hotline: 800-628-8686  
International Hotline: 916-356-7599
- Send e-mail to [developer\\_support@intel.com](mailto:developer_support@intel.com)

## 2. FlashPix Overview

FlashPix\* is a multi-resolution, tiled file format that allows images to be stored at different resolutions for different purposes, such as editing or printing. Each resolution is divided into 64 x 64 pixel tiles. Within a tile, pixels can be either uncompressed, JPEG compressed, or single-color compressed.

FlashPix objects are stored in structured storage container files (OLE Compound Documents) and the image data is stored in defined color spaces.

FlashPix was designed to be a high performance, but highly extensible image container. For more information on FlashPix (including the formal specification), visit the FlashPix Web site at <http://www.kodak.com/US/en/digital/flashPix/>.

### 2.1 FlashPix Storage

In this document the term “stream” is used to mean an OLE Compound Document stream, and the term “storage” is used to mean an OLE Compound Document storage.

Within a resolution, the FlashPix image data is stored in two streams: a Subimage header stream and a Subimage data stream.

- The Subimage header stream determines image data location in the data stream and contains information for decoding the Subimage data stream into uncompressed pixel values. Within the Subimage header stream is a table of Tile Headers that specifies header data for each tile. Namely, a Tile Header contains the location and encoded form of the image data tiles within the second stream.
- The Subimage data stream is a “BLOB” containing each tile’s bit stream of the image typically concatenated together.

The structure of a Tile Header is as follows:

Field Name	Length	Byte(s)
Tile Offset	4	0-3
Tile Size	4	4-7
Compression Type	4	8-11
Compression Sub-Type:	4:	12-15:
Interleave Type	1	12
Chroma Subsampling	1	13
Internal Color Conversion	1	14
JPEG Tables Selector	1	15

**Table 1 - Format and Fields of a Tile Header**

## 3. Decoding JPEG FlashPix Tiles

A JPEG compressed FlashPix tile may be retrieved by:

1. Indexing into the Tile Header table within the subimage header stream.
2. Get a tile's Tile Header.
3. Use the "Size" and "Offset" fields to index into the subimage data stream.
4. Copy the tile's compressed image data (i.e., the JPEG bit stream) into a temporary buffer.
5. Determine where to get the JPEG tables for this tile (i.e., embedded in the tile data or from the global JPEG tables).
6. Use the IJL to decompress the tile.

If a tile is compressed using JPEG, the JPEG Tables Selector subfield in the Tile Header determines whether the tile's bit stream contains its JPEG tables (i.e., quantization and Huffman tables) used during decompression, or whether it uses some "global" JPEG tables. The format for the tile's compressed image data (i.e., the JPEG bit stream) conforms to the JPEG Abbreviated Format for compressed image data.

The function of the JPEG Tables Selector subfield in the Tile Header is to select a set of quantization and Huffman tables to use to decompress the tile. If the JPEG Tables Selector subfield is zero (0x00), then the tables are included at the beginning of the tile's subimage data stream and it is not necessary to load a separate data stream to decompress the tile. However, a non-zero subfield means that the tile's bit stream does not contain embedded JPEG tables, and the non-zero value will be a number (0x01–0xFF) corresponding to the set of JPEG tables needs that has already been globally stored.

The global JPEG tables can be referenced in the Compression Description Group within the Image Contents Property Set of the FlashPix Image Object Storage. The Compression Description Group contains set(s) of JPEG tables that are used across all resolutions of the FlashPix image. The format of the data in each of the JPEG tables conforms to the JPEG Abbreviated Format for table specification data. By storing multiple JPEG abbreviated header tables, different sets of tables may be used by different tiles and may utilize the same table.

### 3.1 Example

Working assumptions for the example:

- The tile is in an addressable memory region (it could be obtained either by actually copying the data or by memory-mapping a portion of the FlashPix file).
- The data is decoded from the compressed tile into a memory region designed to hold exactly one uncompressed tile.

- The Compression Type field in the Tile Header has a value of 0x02, which is 8-bit JPEG compression. The Interleave Type and Chroma Subsampling fields are redundant with information contained in the JPEG tile itself and are ignored.

If the JPEG tables are contained within the compressed FlashPix tile, it is easy to use the IJL to decompress the tile.

```
//-----  
// An example using the Intel® JPEG Library:  
// -- Read a JPEG compressed FlashPix tile  
//   assuming that the JPEG tables are contained  
//   within the FlashPix tile.  
//-----  
  
#include "ijl.h"  
  
struct TileHeader {  
    // Assumes long = 32 bits and unsigned char = 8 bits.  
    long tileOffset;  
    long tileSize;  
    long compressionType;  
    unsigned char interleaveType;  
    unsigned char chromaSubsampling;  
    unsigned char internalColorConversion;  
    unsigned char JPEGTablesSelector;  
}  
  
void Decompress_FlashPix_Tile(  
    const TileHeader &tileIdent,  
    void *rawtile,  
    unsigned char *decompressed_tile)  
{  
    JPEG_CORE_PROPERTIES jcprops;  
  
    // Return immediately if the tile is not compressed using JPEG.  
    // In a more robust implementation, the other compression  
    // types would also be handled.  
    if (tileIdent.compressionType != 0x02)  
        return;  
  
    if (JPEGTablesSelector == 0x00) {  
        // The JPEG tables are embedded in the tile's bit stream.  
  
        // Initialize the Intel® JPEG Library.  
        if (ijlInit(&jcprops) != IJL_OK) {  
            ijlFree(&jcprops);  
            return;  
        }  
  
        // Get information on the JPEG image.  
        jcprops.JPGBytes = rawtile;  
        jcprops.JPGSizeBytes = tileIdent.tileSize;  
        if (ijlRead(&jcprops, IJL_JBUFF_READPARAMS) != IJL_OK) {  
            ijlFree(&jcprops);  
            return;  
        }  
    }  
}
```

```

jcprops.DIBBytes = decompressed_tile;
jcprops.DIBWidth = 64; // All FlashPix tiles are 64x64.
jcprops.DIBHeight = 64; // All FlashPix tiles are 64x64.
jcprops.DIBChannels = 4; // Assumed for this example.

// Set the color space info.
if (tileIdent.internalColorConversion == 1) {
    if (jcprops.JPGChannels = 3) {
        jcprops.JPGColor = IJL_YCBCR;
        jcprops.DIBColor = IJL_RGBA_FPX;
    } else if (jcprops.JPGChannels = 4) {
        jcprops.JPGColor = IJL_YCBCRA_FPX;
        jcprops.DIBColor = IJL_RGBA_FPX;
    } else {
        // The resolution color space is unknown, so
        // do not perform any (internal) color twists.
        jcprops.DIBColor = (IJL_COLOR) IJL_OTHER;
        jcprops.JPGColor = (IJL_COLOR) IJL_OTHER;
    }
}
else {
    // The resolution color space is unknown, so
    // do not perform any (internal) color twists.
    jcprops.DIBColor = (IJL_COLOR) IJL_OTHER;
    jcprops.JPGColor = (IJL_COLOR) IJL_OTHER;
}

// Read the data to the decompressed tile.
if (ijlRead(&jcprops, IJL_JBUFF_READWHOLEIMAGE) != IJL_OK) {
    ijlFree(&jcprops);
    return;
}

// Clean up the Intel® JPEG Library.
if (ijlFree(&jcprops) != IJL_OK)
    return;
}
} // end of Decompress_FlashPix_Tile

```

If the JPEG tables are NOT contained within the tile's bit stream, then they first need to be obtained, placed into the required format for the IJL decoder, and stored for later use. Typically in FlashPix files, only one set of JPEG tables is used for the entire resolution. It is then important to avoid having to reformat the tables for each tile.

Thus, a table of formatted JPEG tables associated with each resolution in the FlashPix root storage is retained. Whenever a Tile Header indicates that a set of JPEG tables is needed that has not been previously accessed, the JPEG tables are decoded, formatted, and persisted.

Before decoding a tile's bit stream that does not contain JPEG tables, a copy of the formatted JPEG tables gets placed into the JPEG\_PROPERTIES data structure. (Note, copying the persisted tables to JPEG\_PROPERTIES is typically much faster than appending a table stream to the front of each JPEG data stream and forcing the decoder

to process and format the tables at every call.) To accomplish this, a new function is added to the example that takes compressed data in the JPEG Abbreviated Format for table specifications, decodes it, and writes the formatted results to a globally persisted structure.

```
//-----
// An example using the Intel® JPEG Library:
// -- Read a JPEG bit stream in the Abbreviated Format
//    for table specification data and persist the formatted
//    tables for later use.
//-----

// This is a prototype for the persisted JPEG tables structure.
// The FlashPix decoder will probably keep up to 255 of these tables.
struct PersistTables {
    int isvalid;
    int nqtables;
    int nhufftables;
    HUFFMAN_TABLES dcHuffmanTable[4];
    HUFFMAN_TABLES acHuffmanTable[4];
    QUANT_TABLE quantTable[4];
}

void Format_JPEG_Tables(
    void *JPEGHeader,
    long headerSize,
    PersistTables *persistTables)
{
    JPEG_CORE_PROPERTIES jccprops;

    // Initialize the Intel® JPEG Library.
    if (ijlInit(&jccprops) != IJL_OK) {
        ijlFree(&jccprops);
        return;
    }

    // Set the JPEG image.
    jccprops.JPGBytes = JPEGHeader;
    jccprops.JPGSizeBytes = headerSize;

    // Decode and format the JPEG tables.
    // After this, the JPEG tables are located in:
    //   jccprops.jprops.jFmtDcHuffman,
    //   jccprops.jprops.jFmtAcHuffman, and
    //   jccprops.jprops.jFmtQuant.
    // There will be a total of:
    //   jccprops.jprops.nhufftables Huffman tables, and
    //   jccprops.jprops.nqtables quantization tables.
    if (ijlRead(&jccprops, IJL_JBUFF_READHEADER) != IJL_OK) {
        ijlFree(&jccprops);
        return;
    }

    // Write the JPEG tables to the persisted storage.
    persistTables->isvalid = true;
    persistTables->nqtables = jccprops.jprops.nqtables;
    persistTables->nhufftables = jccprops.jprops.nhufftables;
}
```

```

// Write the Huffman tables.
for (int i=0; i < jcprops.jprops.nhufftables; i++) {
    memcpy(persistTables->dcHuffmanTable[i],
           jcprops.jprops.jFmtDcHuffman[i],
           sizeof(HUFFMAN_TABLES));

    memcpy(persistTables->acHuffmanTable[i],
           jcprops.jprops.jFmtAcHuffman[i],
           sizeof(HUFFMAN_TABLES));
}

// Write the quantization tables.
for (i=0; i < jcprops.jprops.nqtables; i++) {
    persistTables->quantTable[i].ident =
        jcprops.jprops.jFmtQuant[i].ident;

    persistTables->quantTable[i].precision =
        jcprops.jprops.jFmtQuant[i].precision;

    // Quad-word align the persisted pointer.
    persistTables->quantTable[i].elements =
        (short *) (((int) persistTables->quantTable[i].elarray +
                    0x07) & 0xFFFFFFFF8);

    for (int index=0; index < 77; index++) {
        persistTables->quantTable[i].elements[index] =
            jcprops.jprops.jFmtQuant[i].elements[index];
    }
}

// Clean up the Intel® JPEG Library.
if (ijlFree(&jcprops) != IJL_OK)
    return;

} // end of Format_JPEG_Tables

```

And a new function to copy the persisted tables back into the JPEG\_PROPERTIES structure:

```

//-----
// An example using the Intel® JPEG Library:
// -- Load JPEG tables from persisted storage.
//-----

void Load_Persisted_JPEG_Tables(
    PersistTables *persistTables,
    JPEG_CORE_PROPERTIES *jcprops)
{
    for (int i=0; i<persistTables->nhufftables; i++) {
        memcpy(jcprops.jprops.jFmtDcHuffman[i],
               persistTables->dcHuffmanTable[i],
               sizeof(HUFFMAN_TABLES));

        memcpy(jcprops.jprops.jFmtAcHuffman[i],

```

```

        sizeof(HUFFMAN_TABLES));
    }

    for (int i=0; i < jpegTables->nqttables; i++) {
        jcprops.jprops.jFmtQuant[i].ident =
            persistTables->quantTable[i].ident;
        jcprops.jprops.jFmtQuant[i].precision =
            persistTables->quantTable[i].precision;

        // Quad-word align the persisted pointer.
        jcprops.jprops.jFmtQuant[i].element =
            (short *) (((int) jcprops.jprops.jFmtQuant[i].elarray +
                0x07) & 0xFFFFFFFF8);

        for (int index=0; index < 77; index++) {
            jcprops.jprops.jFmtQuant[i].elements[index] =
                persistTables->quantTable[i].elements[index];
        }
    }
} // end of Load_Persisted_JPEG_Tables

```

The function `Decompress_FlashPix_Tile( )` can be updated to properly handle these new requirements. A few parameters are added:

- `persistTables`, is an STL map of the formatted JPEG tables that get persisted.
- `JPEGTables`, a map of the raw JPEG Headers probably obtained from the FlashPix Compression Property Group
- `JPEGSizes`, a map of the size of each of these headers.
- The indices on each of these maps have the same meaning, and would correspond to the actual index obtained in the Tile Header.

Thus, the final version of `Decompress_FlashPix_Tile( )` is as follows:

```

//-----
// An example using the Intel® JPEG Library:
// -- Read a JPEG compressed FlashPix tile.
//-----

#include <map>
#include "ijl.h"

struct TileHeader {
    // Assumes long = 32 bits and unsigned char = 8 bits.
    long tileOffset;
    long tileSize;
    long compressionType;
    unsigned char interleaveType;
    unsigned char chromaSubsampling;
}

```

```

    unsigned char JPEGTablesSelector;
}

void Decompress_FlashPix_Tile(
    const TileHeader &tileIdent,
    void *rawtile,
    unsigned char *decompressed_tile,
    map<int, PersistTables> &persistTables,
    map<int, void *> &JPEGTables,
    map<int, long> &JPEGSizes)
{
    JPEG_CORE_PROPERTIES jcprops;

    // Return immediately if the tile is not compressed using JPEG.
    // In a more robust implementation, the other compression
    // types would also be handled.
    if (tileIdent.compressionType != 0x02)
        return;

    // Initialize the Intel® JPEG Library.
    if (ijlInit(&jcprops) != IJL_OK) {
        ijlFree(&jcprops);
        return;
    }

    if (tileIdent.JPEGTablesSelector != 0x00)
    {
        // The JPEG tables are NOT embedded in the tile's bit stream.

        PersistTables *jpegTables =
            &persistTables[tileIdent.JPEGTableSelector];

        if (!jpegTables->isvalid)
        {
            // Note: This function will be called infrequently
            // for most FlashPix images.
            Format_JPEG_Tables(
                JPEGTables[tileIdent.JPEGTableSelector],
                JPEGSizes[tileIdent.JPEGTableSelector],
                jpegTables);
        }

        // Copy the Persisted JPEG Tables into the JPEG_PROPERTIES
        // data structure. This function will be called nearly
        // every tile on most FlashPix images.
        Load_Persisted_JPEG_Tables(jpegTables, &jcprops);
    }

    // Get information on the JPEG image.
    jcprops.JPGBytes = rawtile;
    jcprops.JPGSizeBytes = tileIdent.tileSize;
    if (ijlRead(&jcprops, IJL_JBUFF_READPARAMS) != IJL_OK) {
        ijlFree(&jcprops);
        return;
    }

    // Set the DIB data information.

```

```
jcprops.DIBWidth = 64; // All FlashPix tiles are 64x64.
jcprops.DIBHeight = 64; // All FlashPix tiles are 64x64.
jcprops.DIBChannels = 4; // Assumed for this example.

// Set the color space info.
if (tileIdent.internalColorConversion == 1) {
    if (jcprops.JPGChannels = 3) {
        jcprops.JPGColor = IJL_YCBCR;
        jcprops.DIBColor = IJL_RGBA_FPX;
    } else if (jcprops.JPGChannels = 4) {
        jcprops.JPGColor = IJL_YCBCRA_FPX;
        jcprops.DIBColor = IJL_RGBA_FPX;
    } else {
        // The resolution color space is unknown, so
        // do not perform any (internal) color twists.
        jcprops.DIBColor = (IJL_COLOR) IJL_OTHER;
        jcprops.JPGColor = (IJL_COLOR) IJL_OTHER;
    }
}
else {
    // The resolution color space is unknown, so
    // do not perform any (internal) color twists.
    jcprops.DIBColor = (IJL_COLOR) IJL_OTHER;
    jcprops.JPGColor = (IJL_COLOR) IJL_OTHER;
}

// Read the data to the decompressed tile.
if (ijlRead(&jcprops, IJL_JBUFF_READWHOLEIMAGE) != IJL_OK) {
    ijlFree(&jcprops);
    return;
}

// Clean up the Intel® JPEG Library.
if (ijlFree(&jcprops) != IJL_OK)
    return;
} // end of Decompress_FlashPix_Tile
```

## 4. Authoring JPEG FlashPix Tiles

Typically authoring of FlashPix files is simpler than decoding because an authoring engine needs to implement support for only one FlashPix type.

### 4.1 Example

The next example assumes the user is starting from some uncompressed, 4-channel RGBA color space with pre-multiplied opacity and wants to author FlashPix tiles with internal color conversion to the 4-channel FlashPix YCbCrA color space with 4:1:1:4 subsampling. The steps to be taken are:

1. Specify how JPEG tables are stored efficiently.
2. Perform the encoding from an uncompressed FlashPix RGBA tile to a JPEG encoded FlashPix YCbCrA tile.
3. Set the Tile Header appropriately.

For maximum interoperability, the “standard” JPEG tables are used for the encoding process. These are the default tables available from the IJL and recommended by the JPEG specification.

JPEG tables setup:

```
//-----
// An example using the Intel® JPEG Library:
// -- Write out the JPEG Tables in the Abbreviated
//    Format for table specifications.
//-----

#include "ijl.h"

void Write_JPEG_Tables(
    void *jpegHeader,
    long &tileSize)
{
    JPEG_CORE_PROPERTIES jcprops;

    // Initialize the Intel® JPEG Library.
    if (ijlInit(&jcprops) != IJL_OK) {
        ijlFree(&jcprops);
        return;
    }

    // Set the JPEG destination to the input
    // jpegHeader parameter.  The size of the
    // header will be fairly small, typically
    // only a couple hundred bytes.
    // Here, 2K is used as a convenient size.
    jcprops.JPGBytes = jpegHeader;
    jcprops.JPGSizeBytes = 0x800;
}
```

```

    if (ijlWrite(&jcprops, IJL_JBUFF_WRITEHEADER) != IJL_OK) {
        ijlFree(&jcprops);
        return;
    }

    // The actual size of the written JPEG stream is now available.
    tilesize = jcprops.JPGSizeBytes;

    // Clean up the Intel® JPEG Library.
    if (ijlFree(&jcprops) != IJL_OK)
        return;

} // end of Write_JPEG_Tables

```

Writing a tile is much simpler because the built-in IJL entropy tables may be used, and persistence is not an issue like in the decoder:

```

//-----
// An example using the Intel® JPEG Library:
// -- Write out a JPEG compressed FlashPix tile
//-----

#include "ijl.h"

void Compress_FlashPix_Tile(
    unsigned char *decompressed_tile,
    void *rawtile,
    long &rawsize,
    const TileIdentifier &tileIdentifier)
{
    JPEG_CORE_PROPERTIES jcprops;

    // Initialize the Intel® JPEG Library.
    if (ijlInit(&jcprops) != IJL_OK) {
        ijlFree(&jcprops);
        return;
    }

    // Configure the input DIB, which we know a priori
    // to be a 64x64 pixel DIB (no padding), 4 channels,
    // and in the RGBA color space.
    jcprops.DIBBytes = decompressed_tile;
    jcprops.DIBWidth = 64;
    jcprops.DIBHeight = 64;
    jcprops.DIBChannels = 4;
    jcprops.DIBColor = IJL_RGBA_FPX;

    // Configure the output JPEG bit stream to be in the
    // IJL_YCBCRA_FPX color space.
    jcprops.JPGBytes = rawtile;
    jcprops.JPGColor = IJL_YCBCRA_FPX;
    jcprops.JPGSizeBytes = 0x8000; // The size will be <32K.

    // Write the entropy information

```

```
        ijlFree(&jcprops);
        return;
    }

    // The actual size of the written JPEG stream is now available.
    rawsize = jcprops.jprops.JPGSizeBytes;

    // Clean up the Intel® JPEG Library.
    if (ijlFree(&jcprops) != IJL_OK)
        return;

} // end of Compress_FlashPix_Tile
```

Finally, at some point the Tile Header table needs to be written out, which requires a Tile Header.

```
//-----
// An example using the Intel® JPEG Library:
// -- Write out a Tile Header table.
//-----

void Set_Tile_Header(
    TileHeader &tileIdentifier,
    long size,
    long offset)
{
    // Set the tile identifier using the supplied
    // offset and size information for the raw size
    // of the tile. The other parameters are known
    // a priori.

    tileIdentifier.tileOffset = offset;

    tileIdentifier.tileSize = size;

    // JPEG compressed.
    tileIdentifier.compressionType = 0x02;

    tileIdentifier.interleaveType = 0x00;

    // 4:1:1:4 subsampling.
    tileIdentifier.chromaSubsampling = 0x22;

    // Color conversion performed to FlashPix YCbCrA.
    tileIdentifier.internalColorConversion = 0x01;

    // Use the 1st & only set of JPEG tables.
    tileIdentifier.JPEGTablesSelector = 0x01;

} // end of Set_Tile_Header
```