Network Working Group
Request for Comments: 2035
Category: Standards Track

L. Berc
Digital Equipment Corporation
W. Fenner
Xerox PARC
R. Frederick
Xerox PARC
S. McCanne
Lawrence Berkeley Laboratory

October 1996

RTP Payload Format for JPEG-compressed Video

Status of this Memo

Abstract

This memo describes the RTP payload format for JPEG video streams. The packet format is optimized for real-time video streams where codec parameters change rarely from frame to frame.

This document is a product of the Audio-Video Transport working group within the Internet Engineering Task Force. Comments are solicited and should be addressed to the working group's mailing list at rem-conf@es.net and/or the author(s).

1. Introduction

The Joint Photographic Experts Group (JPEG) standard [1,2,3] defines a family of compression algorithms for continuous-tone, still images. This still image compression standard can be applied to video by compressing each frame of video as an independent still image and transmitting them in series. Video coded in this fashion is often called Motion-JPEG.

We first give an overview of JPEG and then describe the specific subset of JPEG that is supported in RTP and the mechanism by which JPEG frames are carried as RTP payloads.

The JPEG standard defines four modes of operation: the sequential DCT mode, the progressive DCT mode, the lossless mode, and the hierarchical mode. Depending on the mode, the image is represented

in one or more passes. Each pass (called a frame in the JPEG standard) is further broken down into one or more scans. Within each scan, there are one to four components,which represent the three components of a color signal (e.g., "red, green, and blue", or a luminance signal and two chromanince signals). These components can be encoded as separate scans or interleaved into a single scan.

Each frame and scan is preceded with a header containing optional definitions for compression parameters like quantization tables and Huffman coding tables. The headers and optional parameters are identified with "markers" and comprise a marker segment; each scan appears as an entropy-coded bit stream within two marker segments. Markers are aligned to byte boundaries and (in general) cannot appear in the entropy-coded segment, allowing scan boundaries to be determined without parsing the bit stream.

Compressed data is represented in one of three formats: the interchange format, the abbreviated format, or the table-specification format. The interchange format contains definitions for all the table used in the by the entropy-coded segments, while the abbreviated format might omit some assuming they were defined out-of-band or by a "previous" image.

The JPEG standard does not define the meaning or format of the components that comprise the image. Attributes like the color space and pixel aspect ratio must be specified out-of-band with respect to the JPEG bit stream. The JPEG File Interchange Format (JFIF) [4] is a defacto standard that provides this extra information using an application marker segment (APP0). Note that a JFIF file is simply a JPEG interchange format image along with the APP0 segment. In the case of video, additional parameters must be defined out-of-band (e.g., frame rate, interlaced vs. non-interlaced, etc.).

While the JPEG standard provides a rich set of algorithms for flexible compression, cost-effective hardware implementations of the full standard have not appeared. Instead, most hardware JPEG video codecs implement only a subset of the sequential DCT mode of operation. Typically, marker segments are interpreted in software (which "re-programs" the hardware) and the hardware is presented with a single, interleaved entropy-coded scan represented in the YUV color space.

2. JPEG Over RTP

To maximize interoperability among hardware-based codecs, we assume the sequential DCT operating mode [1,Annex F] and restrict the set of predefined RTP/JPEG "type codes" (defined below) to single-scan, interleaved images. While this is more restrictive than even

baseline JPEG, many hardware implementation fall short of the baseline specification (e.g., most hardware cannot decode non-interleaved scans).

In practice, most of the table-specification data rarely changes from frame to frame within a single video stream. Therefore, RTP/JPEG data is represented in abbreviated format, with all of the tables omitted from the bit stream. Each image begins immediately with the (single) entropy-coded scan. The information that would otherwise be in both the frame and scan headers is represented entirely within a 64-bit RTP/JPEG header (defined below) that lies between the RTP header and the JPEG scan and is present in every packet.

While parameters like Huffman tables and color space are likely to remain fixed for the lifetime of the video stream, other parameters should be allowed to vary, notably the quantization tables and image size (e.g., to implement rate-adaptive transmission or allow a user to adjust the "quality level" or resolution manually). Thus explicit fields in the RTP/JPEG header are allocated to represent this information. Since only a small set of quantization tables are typically used, we encode the entire set of quantization tables in a small integer field. The image width and height are encoded explicitly.

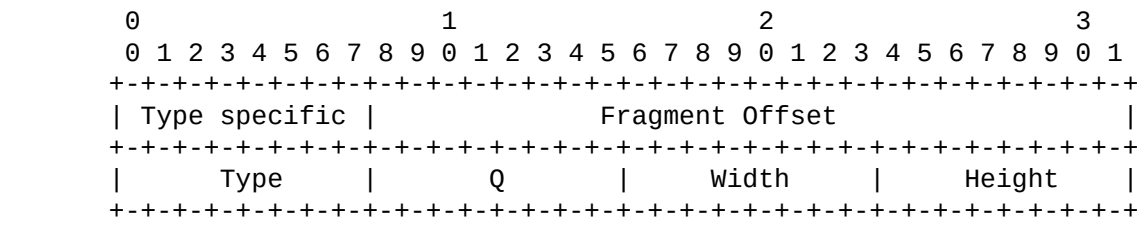
Because JPEG frames are typically larger than the underlying network's maximum packet size, frames must often be fragmented into several packets. One approach is to allow the network layer below RTP (e.g., IP) to perform the fragmentation. However, this precludes rate-controlling the resulting packet stream or partial delivery in the presence of loss. For example, IP will not deliver a fragmented datagram to the application if one or more fragments is lost, or IP might fragment an 8000 byte frame into a burst of 8 back-to-back packets. Instead, RTP/JPEG defines a simple fragmentation and reassembly scheme at the RTP level.

3. RTP/JPEG Packet Format

The RTP timestamp is in units of 90000Hz. The same timestamp must appear across all fragments of a single frame. The RTP marker bit is set in the last packet of a frame.

3.1. JPEG header

A special header is added to each packet that immediately follows the RTP header:



3.1.1. Type specific: 8 bits

Interpretation depends on the value of the type field.

3.1.2. Fragment Offset: 24 bits

The Fragment Offset is the data offset in bytes of the current packet in the JPEG scan.

3.1.3. Type: 8 bits

The type field specifies the information that would otherwise be present in a JPEG abbreviated table-specification as well as the additional JFIF-style parameters not defined by JPEG. Types 0-127 are reserved as fixed, well-known mappings to be defined by this document and future revisions of this document. Types 128-255 are free to be dynamically defined by a session setup protocol (which is beyond the scope of this document).

3.1.4. Q: 8 bits

The Q field defines the quantization tables for this frame using an algorithm that determined by the Type field (see below).

3.1.5. Width: 8 bits

This field encodes the width of the image in 8-pixel multiples (e.g., a width of 40 denotes an image 320 pixels wide).

3.1.6. Height: 8 bits

This field encodes the height of the image in 8-pixel multiples (e.g., a height of 30 denotes an image 240 pixels tall).

3.1.7. Data

The data following the RTP/JPEG header is an entropy-coded segment consisting of a single scan. The scan header is not present and is inferred from the RTP/JPEG header. The scan is terminated either implicitly (i.e., the point at which the image is fully parsed), or explicitly with an EOI marker. The scan may be padded to arbitrary length with undefined bytes. (Existing hardware codecs generate extra lines at the bottom of a video frame and removal of these lines would require a Huffman-decoding pass over the data.)

As defined by JPEG, restart markers are the only type of marker that may appear embedded in the entropy-coded segment. The "type code" determines whether a restart interval is defined, and therefore whether restart markers may be present. It also determines if the restart intervals will be aligned with RTP packets, allowing for partial decode of frames, thus increasing resiliance to packet drop. If restart markers are present, the 6-byte DRI segment (define restart interval marker [1, Sec. B.2.4.4] precedes the scan).

JPEG markers appear explicitly on byte aligned boundaries beginning with an 0xFF. A "stuffed" 0x00 byte follows any 0xFF byte generated by the entropy coder [1, Sec. B.1.1.5].

4. Discussion

4.1. The Type Field

The Type field defines the abbreviated table-specification and additional JFIF-style parameters not defined by JPEG, since they are not present in the body of the transmitted JPEG data. The Type field must remain constant for the duration of a session.

Six type codes are currently defined. They correspond to an abbreviated table-specification indicating the "Baseline DCT sequential" mode, 8-bit samples, square pixels, three components in the YUV color space, standard Huffman tables as defined in [1, Annex K.3], and a single interleaved scan with a scan component selector indicating components 0, 1, and 2 in that order. The Y, U, and V color planes correspond to component numbers 0, 1, and 2, respectively. Component 0 (i.e., the luminance plane) uses Huffman table number 0 and quantization table number 0 (defined below) and components 1 and 2 (i.e., the chrominance planes) use Huffman table number 1 and quantization table number 1 (defined below).

Additionally, video is non-interlaced and unscaled (i.e., the aspect ratio is determined by the image width and height). The frame rate is variable and explicit via the RTP timestamp.

Six RTP/JPEG types are currently defined that assume all of the above. The odd types have different JPEG sampling factors from the even ones:

types	comp	horizontal samp. fact.	vertical samp. fact.
0/2/4	0	2	1
0/2/4	1	1	1
0/2/4	2	1	1
1/3/5	0	2	2
1/3/5	1	1	1
1/3/5	2	1	1

These sampling factors indicate that the chromanince components of type 0/2/4 video is downsampled horizontally by 2 (often called 4:2:2) while the chrominance components of type 1/3/5 video are downsampled both horizontally and vertically by 2 (often called 4:2:0).

The three pairs of types (0/1), (2/3) and (4/5) differ from each other as follows:

0/1 : No restart markers are present in the entropy data.
No restriction is placed on the fragmentation of the stream into RTP packets.
The type specific field is unused and must be zero.

2/3 : Restart markers are present in the entropy data.
The entropy data is preceded by a DRI marker segment, defining the restart interval.
No restriction is placed on the fragmentation of the stream into RTP packets.
The type specific field is unused and must be zero.

4/5 : Restart markers are present in the entropy data.
The entropy data is preceded by a DRI marker segment, defining the restart interval.
Restart intervals are be sent as separate (possibly multiple) RTP packets.
The type specific field (TSPEC) is used as follows:
A restart interval count (RCOUNT) is defined, which starts at zero, and is incremented for each restart interval in the frame.

The first packet of a restart interval gets TSPEC = RCOUNT. Subsequent packets of the restart interval get TSPEC = 254, except the final packet, which gets TSPEC = 255.

Additional types in the range 128-255 may be defined by external means, such as a session protocol.

Appendix B contains C source code for transforming the RTP/JPEG header parameters into the JPEG frame and scan headers that are absent from the data payload.

4.2. The Q Field

The quantization tables used in the decoding process are algorithmically derived from the Q field. The algorithm used depends on the type field but only one algorithm is currently defined for the two types.

Both type 0 and type 1 JPEG assume two quantizations tables. These tables are chosen as follows. For 1 <= Q <= 99, the Independent JPEG Group's formula [5] is used to produce a scale factor S as:

```
S = 5000 / Q          for  1 <= Q <= 50
  = 200 - 2 * Q       for 51 <= Q <= 99
```

This value is then used to scale Tables K.1 and K.2 from [1] (saturating each value to 8-bits) to give quantization table numbers 0 and 1, respectively. C source code is provided in Appendix A to compute these tables.

For Q >= 100, a dynamically defined quantization table is used, which might be specified by a session setup protocol. (This session protocol is beyond the scope of this document). It is expected that the standard quantization tables will handle most cases in practice, and dynamic tables will be used rarely. Q = 0 is reserved.

4.3. Fragmentation and Reassembly

Since JPEG frames are large, they must often be fragmented. Frames should be fragmented into packets in a manner avoiding fragmentation at a lower level. When using restart markers, frames should be fragmented such that each packet starts with a restart interval (see below).

Each packet that makes up a single frame has the same timestamp. The fragment offset field is set to the byte offset of this packet within the original frame. The RTP marker bit is set on the last packet in a frame.

An entire frame can be identified as a sequence of packets beginning with a packet having a zero fragment offset and ending with a packet having the RTP marker bit set. Missing packets can be detected either with RTP sequence numbers or with the fragment offset and lengths of each packet. Reassembly could be carried out without the offset field (i.e., using only the RTP marker bit and sequence numbers), but an efficient single-copy implementation would not otherwise be possible in the presence of misordered packets. Moreover, if the last packet of the previous frame (containing the marker bit) were dropped, then a receiver could not detect that the current frame is entirely intact.

4.4. Restart Markers

Restart markers indicate a point in the JPEG stream at which the Huffman codec and DC predictors are reset, allowing partial decoding starting at that point. The use of restart markers allows for robustness in the face of packet loss.

RTP/JPEG Types 4/5 allow for partial decode of frames, due to the alignment of restart intervals with RTP packets. The decoder knows it has a whole restart interval when it gets sequence of packets with contiguous RTP sequence numbers, starting with TSPEC<254 (RCOUNT) and either ending with TSPEC==255, or TSPEC<255 and next packet's TSPEC<254 (or end of frame).

It can then decompress the RST interval, and paint it. The X and Y tile offsets of the first MCU in the interval are given by:

```
tile_offset = RCOUNT * restart_interval * 2
x_offset    = tile_offset % frame_width_in_tiles
y_offset    = tile_offset / frame_width_in_tiles
```

The MCUs in a restart interval may span multiple tile rows.

Decoders can, however, treat types 4/5 as types 2/3, simply reassembling the entire frame and then decoding.

5. Security Considerations

Security issues are not discussed in this memo.

6. Authors' Addresses

Lance M. Berc
Systems Research Center
Digital Equipment Corporation
130 Lytton Ave
Palo Alto CA 94301

Phone: +1 415 853 2100
EMail: berc@pa.dec.com

William C. Fenner
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

Phone: +1 415 812 4816
EMail: fenner@cmf.nrl.navy.mil

Ron Frederick
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

Phone: +1 415 812 4459
EMail: frederick@parc.xerox.com

Steven McCanne
Lawrence Berkeley Laboratory
M/S 46A-1123
One Cyclotron Road
Berkeley, CA 94720

Phone: +1 510 486 7520
EMail: mccanne@ee.lbl.gov

7. References

[1] ISO DIS 10918-1. Digital Compression and Coding of Continuous-tone Still Images (JPEG), CCITT Recommendation T.81.

[2] William B. Pennebaker, Joan L. Mitchell, JPEG: Still Image Data Compression Standard, Van Nostrand Reinhold, 1993.

[3] Gregory K. Wallace, The JPEG Sill Picture Compression Standard, Communications of the ACM, April 1991, Vol 34, No. 1, pp. 31-44.

[4] The JPEG File Interchange Format. Maintained by C-Cube Microsystems, Inc., and available in ftp://ftp.uu.net/graphics/jpeg/jfif.ps.gz.

[5] Tom Lane et. al., The Independent JPEG Group software JPEG codec. Source code available in ftp://ftp.uu.net/graphics/jpeg/jpegsrc.v5.tar.gz.

Appendix A

The following code can be used to create a quantization table from a Q factor:

```
/*
 * Table K.1 from JPEG spec.
 */
static const int jpeg_luma_quantizer[64] = {
        16, 11, 10, 16, 24, 40, 51, 61,
        12, 12, 14, 19, 26, 58, 60, 55,
        14, 13, 16, 24, 40, 57, 69, 56,
        14, 17, 22, 29, 51, 87, 80, 62,
        18, 22, 37, 56, 68, 109, 103, 77,
        24, 35, 55, 64, 81, 104, 113, 92,
        49, 64, 78, 87, 103, 121, 120, 101,
        72, 92, 95, 98, 112, 100, 103, 99
};

/*
 * Table K.2 from JPEG spec.
 */
static const int jpeg_chroma_quantizer[64] = {
        17, 18, 24, 47, 99, 99, 99, 99,
        18, 21, 26, 66, 99, 99, 99, 99,
        24, 26, 56, 99, 99, 99, 99, 99,
        47, 66, 99, 99, 99, 99, 99, 99,
        99, 99, 99, 99, 99, 99, 99, 99,
        99, 99, 99, 99, 99, 99, 99, 99,
        99, 99, 99, 99, 99, 99, 99, 99,
        99, 99, 99, 99, 99, 99, 99, 99
};

/*
 * Call MakeTables with the Q factor and two int[64] return arrays
 */
void
MakeTables(int q, u_char *lum_q, u_char *chr_q)
{
  int i;
  int factor = q;

  if (q < 1) factor = 1;
  if (q > 99) factor = 99;
  if (q < 50)
    q = 5000 / factor;
  else
    q = 200 - factor*2;
```

```
  for (i=0; i < 64; i++) {
    int lq = ( jpeg_luma_quantizer[i] * q + 50) / 100;
    int cq = ( jpeg_chroma_quantizer[i] * q + 50) / 100;

    /* Limit the quantizers to 1 <= q <= 255 */
    if ( lq < 1) lq = 1;
    else if ( lq > 255) lq = 255;
    lum_q[i] = lq;

    if ( cq < 1) cq = 1;
    else if ( cq > 255) cq = 255;
    chr_q[i] = cq;
  }
}
```

Appendix B

The following routines can be used to create the JPEG marker segments corresponding to the table-specification data that is absent from the RTP/JPEG body.

```
u_char lum_dc_codelens[] = {
        0, 1, 5, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
};

u_char lum_dc_symbols[] = {
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
};

u_char lum_ac_codelens[] = {
        0, 2, 1, 3, 3, 2, 4, 3, 5, 5, 4, 4, 0, 0, 1, 0x7d,
};

u_char lum_ac_symbols[] = {
        0x01, 0x02, 0x03, 0x00, 0x04, 0x11, 0x05, 0x12,
        0x21, 0x31, 0x41, 0x06, 0x13, 0x51, 0x61, 0x07,
        0x22, 0x71, 0x14, 0x32, 0x81, 0x91, 0xa1, 0x08,
        0x23, 0x42, 0xb1, 0xc1, 0x15, 0x52, 0xd1, 0xf0,
        0x24, 0x33, 0x62, 0x72, 0x82, 0x09, 0x0a, 0x16,
        0x17, 0x18, 0x19, 0x1a, 0x25, 0x26, 0x27, 0x28,
        0x29, 0x2a, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
        0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49,
        0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59,
        0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69,
        0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79,
        0x7a, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89,
        0x8a, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98,
        0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
```

```
        0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6,
        0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3, 0xc4, 0xc5,
        0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2, 0xd3, 0xd4,
        0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xe1, 0xe2,
        0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9, 0xea,
        0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,
        0xf9, 0xfa,
};

u_char chm_dc_codelens[] = {
        0, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
};

u_char chm_dc_symbols[] = {
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
};

u_char chm_ac_codelens[] = {
        0, 2, 1, 2, 4, 4, 3, 4, 7, 5, 4, 4, 0, 1, 2, 0x77,
};

u_char chm_ac_symbols[] = {
        0x00, 0x01, 0x02, 0x03, 0x11, 0x04, 0x05, 0x21,
        0x31, 0x06, 0x12, 0x41, 0x51, 0x07, 0x61, 0x71,
        0x13, 0x22, 0x32, 0x81, 0x08, 0x14, 0x42, 0x91,
        0xa1, 0xb1, 0xc1, 0x09, 0x23, 0x33, 0x52, 0xf0,
        0x15, 0x62, 0x72, 0xd1, 0x0a, 0x16, 0x24, 0x34,
        0xe1, 0x25, 0xf1, 0x17, 0x18, 0x19, 0x1a, 0x26,
        0x27, 0x28, 0x29, 0x2a, 0x35, 0x36, 0x37, 0x38,
        0x39, 0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48,
        0x49, 0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58,
        0x59, 0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68,
        0x69, 0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78,
        0x79, 0x7a, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
        0x88, 0x89, 0x8a, 0x92, 0x93, 0x94, 0x95, 0x96,
        0x97, 0x98, 0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5,
        0xa6, 0xa7, 0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4,
        0xb5, 0xb6, 0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3,
        0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2,
        0xd3, 0xd4, 0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda,
        0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9,
        0xea, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,
        0xf9, 0xfa,
};

u_char *
MakeQuantHeader(u_char *p, u_char *qt, int tableNo)
{
```

```
        *p++ = 0xff;
        *p++ = 0xdb;            /* DQT */
        *p++ = 0;               /* length msb */
        *p++ = 67;              /* length lsb */
        *p++ = tableNo;
        memcpy(p, qt, 64);
        return (p + 64);
}

u_char *
MakeHuffmanHeader(u_char *p, u_char *codelens, int ncodes, u_char *symbols,
                  int nsymbols, int tableNo, int tableClass)
{
        *p++ = 0xff;
        *p++ = 0xc4;            /* DHT */
        *p++ = 0;               /* length msb */
        *p++ = 3 + ncodes + nsymbols; /* length lsb */
        *p++ = tableClass << 4 | tableNo;
        memcpy(p, codelens, ncodes);
        p += ncodes;
        memcpy(p, symbols, nsymbols);
        p += nsymbols;
        return (p);
}

/*
 * Given an RTP/JPEG type code, q factor, width, and height,
 * generate a frame and scan headers that can be prepended
 * to the RTP/JPEG data payload to produce a JPEG compressed
 * image in interchange format (except for possible trailing
 * garbage and absence of an EOI marker to terminate the scan).
 */
int MakeHeaders(u_char *p, int type, int q, int w, int h)
{
        u_char *start = p;
        u_char lqt[64];
        u_char cqt[64];

        /* convert from blocks to pixels */
        w <<= 3;
        h <<= 3;

        MakeTables(q, lqt, cqt);

        *p++ = 0xff;
        *p++ = 0xd8;            /* SOI */

        p = MakeQuantHeader(p, lqt, 0);
```

```
        p = MakeQuantHeader(p, cqt, 1);

        p = MakeHuffmanHeader(p, lum_dc_codelens,
                              sizeof(lum_dc_codelens),
                              lum_dc_symbols,
                              sizeof(lum_dc_symbols), 0, 0);
        p = MakeHuffmanHeader(p, lum_ac_codelens,
                              sizeof(lum_ac_codelens),
                              lum_ac_symbols,
                              sizeof(lum_ac_symbols), 0, 1);
        p = MakeHuffmanHeader(p, chm_dc_codelens,
                              sizeof(chm_dc_codelens),
                              chm_dc_symbols,
                              sizeof(chm_dc_symbols), 1, 0);
        p = MakeHuffmanHeader(p, chm_ac_codelens,
                              sizeof(chm_ac_codelens),
                              chm_ac_symbols,
                              sizeof(chm_ac_symbols), 1, 1);

        *p++ = 0xff;
        *p++ = 0xc0;            /* SOF */
        *p++ = 0;               /* length msb */
        *p++ = 17;              /* length lsb */
        *p++ = 8;               /* 8-bit precision */
        *p++ = h >> 8;          /* height msb */
        *p++ = h;               /* height lsb */
        *p++ = w >> 8;          /* width msb */
        *p++ = w;               /* wudth lsb */
        *p++ = 3;               /* number of components */
        *p++ = 0;               /* comp 0 */
        if (type == 0)
                *p++ = 0x21;    /* hsamp = 2, vsamp = 1 */
        else
                *p++ = 0x22;    /* hsamp = 2, vsamp = 2 */
        *p++ = 0;               /* quant table 0 */
        *p++ = 1;               /* comp 1 */
        *p++ = 0x11;            /* hsamp = 1, vsamp = 1 */
        *p++ = 1;               /* quant table 1 */
        *p++ = 2;               /* comp 2 */
        *p++ = 0x11;            /* hsamp = 1, vsamp = 1 */
        *p++ = 1;               /* quant table 1 */

        *p++ = 0xff;
        *p++ = 0xda;            /* SOS */
        *p++ = 0;               /* length msb */
        *p++ = 12;              /* length lsb */
        *p++ = 3;               /* 3 components */
        *p++ = 0;               /* comp 0 */
```

```
        *p++ = 0;               /* huffman table 0 */
        *p++ = 1;               /* comp 1 */
        *p++ = 0x11;            /* huffman table 1 */
        *p++ = 2;               /* comp 2 */
        *p++ = 0x11;            /* huffman table 1 */
        *p++ = 0;               /* first DCT coeff */
        *p++ = 63;              /* last DCT coeff */
        *p++ = 0;               /* sucessive approx. */

        return (p - start);
};
```