Supporting the DCT Filters in PostScript Level 2

Adobe Developer Support

Technical Note #5116

24 November 1992

Contents

Supporting the DCT Filters in PostScript Level 2

1 Introduction

This technical note describes the operation of the **DCTEncode** and **DCTDecode** filters in PostScript Level 2 version 2010 to 2012 interpreters. Its purpose is to describe these two filters for application software writers whose software must interface to PostScript Level 2 interpreters.

It is a supplement to the *PostScript Language Reference Manual, Second Edition*, and nothing in the technical note should conflict with the manual. However, there are several extensions to and explanations of that material. Adobe reserves the right to change any part of this document and any part of the implementation that this document describes without notice. This information is for you to use at your own risk.

2 Purpose of the DCTEncode and DCTDecode Filters

The **DCTEncode** filter compresses one, two, three, or four color continuous tone images, where each sample of each color is specified by an 8-bit value. (A one-color continuous tone image is a gray-scale image.) It is a lossy compressor that does not reproduce the original image exactly, but the trade-off between compression and image fidelity is controllable by parameters of the method. On typical images, 15-to-1 compression is achieved without perceptible impairment and 30-to-1 with little impairment. It is possible to operate in a nearly-lossless mode and still achieve 2-to-1 compression.

The **DCTDecode** filter decodes the compressed image. There are few options for the **DCTDecode** filter because nearly all choices are made at the time of compression.

When a color image is composed of color samples that are 2 or 4 bits, the **DCTEncode** and **DCTDecode** filters cannot be used directly. However, an application could swell the samples to 8 bits and still achieve significant compression. 12-bit color samples are not handled by the method; applications must reduce the data to 8 bits before using the DCT filters. Only interleaved images are handled by these filters.

It would be unusual to use the **DCTEncode** filter in a printer or typesetter while printing a document. Normally, the DCT encoding function is carried out by an application program prior to downloading. However, the **DCTEncode** filter is included in Level 2 printer and typesetter products anyway. Applications that do not otherwise implement the DCT encoding function can use the printer or typesetter to compress an image onto a local disk, for example. This abnormal use of the printer or typesetter is more plausible if the communication channel is high speed and if the printer or typesetter has a high performance controller.

3   Alternative Compression Possibilities

The **LZWEncode** and **LZWDecode** filters can be used to achieve lossless image compression. When used without its pixel prediction feature, LZW averaged only 8% compression over seven original scanned images, but it averaged 40% with pixel prediction. LZW can achieve 8-to-1 compression on many machine-generated images. The version 2010 to 2012 LZW filters are somewhat slower than the DCT filters.

The **CCITTFaxEncode** and **CCITTFaxDecode** filters work on monochrome (1 bit/pixel) images. They are not used with continuous tone images and do not compete with the DCT filters.

Another possibility is to prepare a PostScript language program without compression, run the entire program through an **LZWEncode** filter, and then decode and execute the program on a Level 2 printer. The following example shows this with cascaded use of the **ASCII85Encode** filter:

```
%!
currentfile /ASCII85Decode filter /LZWDecode filter cvx
exec ...ASCII85Encoded LZWEncoded PS program...
```

4   Compatibility with JPEG Specifications

The *PostScript Language Reference Manual, Second Edition* states that the **DCTEncode** and **DCTDecode** filters are compatible with JPEG Revision 8, August 14, 1990. The committee preparing the international standard is *ISO/IEC JTC1/SC2/WG10 Photographic Image Coding*. Prior to the establishment of WG10 in 1990, the committee existed as an ad hoc group, known as the *Joint Photographic Experts Group (JPEG)* of ISO/IEC JTC1/SC2/WG8. Both the committee and the image coding processes it has developed are known informally by the name JPEG.

Our information from members of the JPEG committee is that revisions 8-R8 through 9-R7 and the committee draft 10918-1 (the most recent revision at the time of this writing) are identical in technical content and vary only in the exposition of the material. Furthermore, they believe that the technical

content will not change when the JPEG/WG10 document is finally approved as an international standard. In other words, the Level 2 interpreter implementation of the JPEG standard is compatible with all earlier revisions of the JPEG draft standard from 8-R8 to the CD 10918-1 revision, and it is likely to be compatible with what eventually becomes an international standard.

The earlier public revisions to the JPEG draft were 8-R2 and 8-R5. 8-R5 was technically different in that it used 32-bit Huffman codes and was little-endian, whereas 8-R8 to 9-R7 revisions are limited to 16-bit Huffman codes and are big-endian. The marker codes in the representation also changed between 8-R5 and 8-R8. Other differences might also exist. The DCT filters will not interpret these earlier versions of JPEG.

The Level 2 interpreter **DCTEncode** and **DCTDecode** filters deal with the format called *JPEG Interchange Format* in the 9-R7 draft (see JPEG-9-R7: Working Draft for Development of JPEG CD, 14 February 1991). The Baseline method is supported subject to several limitations; several other parts of the JPEG specification are also supported.

5 JPEG Interchange Format

As mentioned above, the embodiment of JPEG accepted by PostScript Level 2 is the JPEG Interchange Format.

A JPEG Interchange Format compressed image begins with an SOI (start-of-image) marker followed by a number of marker segments that define compressed image parameters. This is followed by the coded body of the compressed image and, finally, by an EOI (end-of-image) marker. Each marker segment begins with a 0xFF byte called a Fill marker. The following byte identifies the kind of marker. Many markers have a two-byte length immediately after the marker identifier. The interpretation of the various markers is discussed in the JPEG 9-R7 or other draft.

In the compressed image body, after the initial markers, Baseline images use Huffman coding. An option is to insert Restart markers periodically. When used, it is possible to edit and replace the section of an image between any two Restart markers without having to reencode any other parts of the compressed image.

There are no default values. Every parameter needed to decode the compressed image is contained in the initial markers of the compressed image.

Color transforms (for example, RGB-to-YCC) commonly used with the JPEG Interchange Format are not specified by the interchange format. They are compatible with the interchange format but separate from it. Color transforms are discussed later in this document.

The JPEG File Interchange Format and the JFIF marker it uses are also commonly used with the JPEG Interchange Format. Like the color transforms, they are compatible with the JPEG Interchange Format but are separate from it. An example that inserts a JFIF marker when using the **DCTEncode** filter is presented later in this document.

6   DCTDecode Filter Summary

The **DCTDecode** filter implementation complies with the *PostScript Language Reference Manual, Second Edition* Level 2 specification, with the additions described in this section. It can be described as follows:

- It will decode any JPEG Baseline DCT or Extended sequential DCT compressed image in the JPEG Interchange Format subject to the following restrictions:

  The image may consist of one scan containing one, two, three, or four colors. Images specifying a larger number of colors, having more than one scan, or having more than one frame will not be decoded.

  Zero-size images (**Columns**=0 or **Rows**=0) are invalid.

  There must be enough RAM available in the PostScript interpreter to support the filter. The RAM requirements are described later.

Note   *For JPEG experts only: The SOF0, SOF1, DHT, RSTm, EOI, SOS, DQT, DRI, and COM markers are properly decoded. APPn (application-specific) markers are skipped over harmlessly except for the Adobe reserved marker described later.*

These markers are not decoded: SOF2-SOF15, DAC, DNL, DHP, EXP, JPGn, TEM, and RESn. If any occurs in a compressed image, it will be rejected. With the exception of DNL, none of these markers is useful in a Baseline DCT or Extended sequential DCT image.

- It will decode any file produced by the **DCTEncode** filter. In particular it will interpret the Adobe application-specific marker code produced by the **DCTEncode** filter. This marker is compatible with but not part of the JPEG Baseline specification. Also, it will perform optional YCC-to-RGB or YCCK-to-CMYK color coordinate transforms that are not part of the JPEG specification.

The following parameters can be included in the (optional) dictionary that is supplied to the **DCTDecode** filter. These parameters do not appear in the *PostScript Language Reference Manual, Second Edition*:

- **Picky** *integer* (default 0). Some Level 2 interpreters will perform a more stringent level of error-checking compressed images if **Picky**=1. This is intended as an application debugging option that can slow decoding and might cause rejection of images that would be successfully decoded in its absence. It also reports as bugs the occurrence of extraneous bytes at the end of JPEG marker segments or MCUs and the replication of FIL (0xFF) markers. (Replicated FIL markers are legal in JPEG but probably unintended in most applications.) **Picky** = 1 should not be used in any shipping product.

- **Relax** *integer* (default 0). **Relax** = 1 is a hack equivalent to using SOF1 as the JPEG start-of-frame marker in the compressed image. It exists because several JPEG implementations with which Adobe cross-tested in 1991 were non-Baseline, non-JPEG, or buggy; but they used SOF0 markers anyway. Specifying **Relax** = 1 will accommodate some of those buggy implementations. In the absence of one of these indications, any non-Baseline usage results in an error.

The Level 2 software implementation permits the following useful extensions to the Baseline standard when the JPEG start-of-frame marker is SOF1 instead of SOF0 (or when **Relax**=1):

- Up to **Colors** unique DC and AC **HuffTables**. Baseline allows no more than two DC and two AC Huffman coding tables to facilitate hardware implementations. This is a standard JPEG extension to Baseline described in 9-R7 F.1.3.

- Sum (HS * VS) > 10. JPEG requires that the sum of the products of the horizontal and vertical sampling parameters over all colors not exceed ten, even for extensions to the Baseline standard. This allows hardware implementations to have on-chip buffering for only ten 8x8 blocks. The current software permits this option, which is not allowed by *JPEG*.

However, non-Baseline options are likely to prevent the use of special hardware on some Level 2 or other application products. The greatest degree of interchangeability with other applications will be afforded by staying strictly Baseline.

The following is a list of *PostScript Language Reference Manual, Second Edition* errata.

- The dictionary argument to the filter creation command is optional for the **DCTDecode** filter.

- All references to YUV and YUVK should be changed to YCC and YCCK, respectively, since YUV refers to an analog television signal and has no place in digital imagery. The default **ColorTransform** referred to as RGB-to-YUV in the *PostScript Language Reference Manual, Second Edition* is used for RGB images conforming to the CCIR Rec. 601-1 standard.

7 DCTEncode Filter Summary

The **DCTEncode** filter also complies with the *PostScript Language Reference Manual, Second Edition* specification with the additions described in this section. It produces a JPEG Baseline compressed image consisting of one frame and one scan with one to four colors. The following errata apply to the arguments passed in the dictionary:

- Wherever the manual uses *array* for arguments in the **DCTEncode** filter dictionary, either an array or a packed array can be used.

- **HSamples** and **VSamples** can optionally be strings instead of arrays or packed arrays. Any extra elements in these arrays or strings are ignored.

- Each quantization table can optionally be a string instead of an array or packed array. Also, when the table is an array or packed array, it might contain not only integers but also real numbers.

- Only the first 2 x **Colors** elements of the **HuffTables** array are used. This array is permitted to be longer without causing an error. Each element of the **HuffTables** array (that is, a Huffman table specification) can be a string, array, or packed array of integers.

- **QFactor** can be 0.0, and its maximum value is presently 1000000.0.

The following parameters can be included in the dictionary supplied to the **DCTEncode** filter. These parameters are not in the *PostScript Language Reference Manual, Second Edition*:

- **NoMarker** *boolean* (default false). If true, the output of the Adobe application-specific JPEG marker (described below) is suppressed.

- **Resync** *integer* (default 0). If a non-zero value N is supplied, the **DCTEncode** filter inserts a DRI marker at the beginning of the coded data and will put JPEG restart marker codes between MCUs of the coded data at the spacing N.

- **Blend** *integer* (default 0). If the maximum sampling value N for any color is greater than the sampling value M for a color, that color will be downsampled before compression. The JPEG specification does not specify how downsampling should be carried out.

If **Blend** = 0 (*Chop* mode), a fast method of downsampling will be used, probably selection of particular samples from the group being down-sampled. If **Blend** =1 (*Blend* mode), downsampling will average or merge sample values to get a better representative value. **Blend** = 1 is a hint that asks **DCTEncode** to strive for smoother downsampling. Its implementation might vary among Level 2 products.

- **Markers** string (default none). The **DCTEncode** filter copies this string literally into the compressed output immediately after the JPEG SOI marker. The Markers string can contain one or more COM or APPn (such as JFIF) JPEG markers.

- **Picky** *integer* (default 0). Same as **DCTDecode** filter.

- **Relax** *integer* (default 0). **Relax** =1 allows **DCTEncode** to use any non-Baseline JPEG extensions that are implemented (currently, Sum(HS * VS) > 10 and separate **HuffTables** for each color, as described earlier for the **DCTDecode** filter). If **Relax** = 0, inadvertent specification of non-Baseline values for an option will result in an error.

The following information is supplementary to the *PostScript Language Reference Manual, Second Edition*, description:

Zero-size images (**Columns** = 0 or **Rows** = 0) are invalid. (A JPEG indefinitely-long image is denoted by **Rows** = 0. The length of such an image is defined by a DNL marker. This JPEG feature is not implemented by **DCTEncode** or **DCTDecode**. **Columns** = 0 is disallowed by JPEG.)

8   DCTDecode Program Example

The following excerpt is from a Level 2 program, which images a DCT and
ASCII base-85 encoded compressed image using the **colorimage** operator:

```
%!PS-Adobe-3.0
%%LanguageLevel: 2
%%Creator: ()
%%CreationDate: ()
%%EndComments
save mark {{10 10 translate 0 rotate 648.0 518.4 scale
720 576 8 [720 0 0 -576 0 576]
currentfile /ASCII85Decode filter dup /A85dec exch def
<</ColorTransform 0>> /DCTDecode filter
false 3 colorimage} stopped {handleerror} if
cleartomark A85dec flushfile}
exec ASCII85Encoded JPEG INTERCHANGE FORMAT COMPRESSED IMAGE HERE
restore showpage
%%EOF
```

ASCII base-85 encoding allows the compressed image, with arbitrary binary
codes, to be downloaded to a printer on any communication channel. In addi-
tion, if an error occurs, the **flushfile** on the ASCII base-85 filter should skip
the residual image and resume execution of the PostScript language program
afterwards. If ASCII base-85 encoding is not used, the image will be 20%
smaller but can then only be downloaded on a channel that can handle binary
data (for example, AppleTalk), or by using the binary communication proto-
col, which Level 2 supports for serial channels.

Here is another example using the **image** operator but without error handling
and without using the ASCII base-85 encoding:

```
%!PS-Adobe-3.0
%%LanguageLevel: 2
%%Creator: ()
%%CreationDate: ()
%%EndComments
/DeviceRGB setcolorspace
126 270 translate % Center image on letter-size page
349 252 scale     % Scale image to original size: 4.847 x 3.500
inches
% Create procedure to decode and image the DCT-encoded data.
% Note that 'exec' is followed by exactly one space character.
{ /Data currentfile /DCTDecode filter def
<< /ImageType 1
   /Width 727
   /Height 525
   /ImageMatrix [727 0 0 -525 0 525]
   /DataSource Data
   /BitsPerComponent 8
   /Decode [0 1 0 1 0 1]
>> image showpage } exec JPEG INTERCHANGE FORMAT COMPRESSED IMAGE HERE
%%EOF
```

Note that there is exactly one blank between **exec** and the JPEG Interchange Format data. Using a return or linefeed instead of a blank is unsafe because some communication protocols will turn it into a 'carriage return-linefeed'. As discussed in section 12, "Bugs and Incompatibilities," this substitution could conceivably cause an **ioerror** in PostScript interpreters of versions 2010 and 2011, though it will succeed in versions 2012 and later.

Here is a third example without error handling, without ASCII base-85 encoding, and without **exec**:

```
%!PS-Adobe-3.0
%%LanguageLevel: 2
%%Creator: ()
%%CreationDate: ()
%%EndComments
/DeviceRGB setcolorspace
126 270 translate % Center image on letter-size page
349 252 scale % Scale image to original size: 4.847 x 3.500 inches


/Data currentfile /DCTDecode filter def
<< /ImageType 1
   /Width 727
   /Height 525
   /ImageMatrix [727 0 0 -525 0 525]
   /DataSource Data
   /BitsPerComponent 8
   /Decode [0 1 0 1 0 1]
>> image JPEG INTERCHANGE FORMAT COMPRESSED IMAGE HERE
showpage
%%EOF
```

This is a tricky example. The PostScript interpreter binds the source of the **DCTDecode** filter to **currentfile**, which at that instant points at **/DCTDecode**. However, decoding filters do not read from the source until their first use, so the **DCTDecode** filter will begin processing its input when the interpreter has processed **image**.

9   DCTEncode Program Example

The following excerpt is from a Level 2 program which compresses a left-to-right, top-to-bottom raster 3-color RGB image using the **DCTEncode** filter, and writes the compressed image on another file:

```
jpeg begin            % Open dictionary containing optional params.
save mark 4 -2 roll
{  /dest exch (w) file def  % Open arg2 as output file
   /src exch (r) file def   % Open arg1 as input file

   /Colors 3 def            % Setup image-specific dictionary params.
   /Columns 512 def
   /Rows 512 def

   /buf Columns Colors mul string def
   /filtdest dest jpeg /DCTEncode filter def
   Rows {filtdest src buf readstring pop writestring} repeat
   filtdest closefile dest closefile
} stopped {handleerror} if cleartomark restore
end
```

The default **handleerror** procedure handles all filter error messages adequately.

10   Error Handling

For the **DCTDecode** filter, the Level 2 interpreter requires that no compressed data be read until first use of the filter (because the image often will be coming from **currentfile**, and the compressed image's starting point within **currentfile** is unknown when the filter is created). This means that errors associated with bad parameters in the initial marker segments of the compressed image will occur when the first byte is read rather than when the filter is created and will typically manifest as an **ioerror** for the operator using the filter.

For the **DCTEncode** filter, most errors associated with bad parameters in the argument dictionary are detected during filter creation. These include many **rangecheck** and **typecheck** conditions and a **limitcheck** (not enough storage) for the **filter** operator. Some **HuffTables** error checks occur when the first byte is written to the filter. If one of these errors occurred, it would be reported as an **ioerror** for the operator using the **filter**.

Many different errors can also occur during image encoding or decoding. Because the number of error conditions that can occur during filter operation is large and complex, both filters utilize the **errorinfo** entry of the **$error** dictionary to record ancillary information about each particular error. This is discussed in section 3.10.2, "Error Handling," in the *PostScript Language Reference Manual, Second Edition*, although the section does not indicate

that **errorinfo** is used by filters. In some cases, the **errorinfo** report consists of a key name in a dictionary (for example, **HuffTables** or **Colors**) followed by an illegal value.

In other cases, the information recorded in **errorinfo** consists of two strings, the name of the filter and an error message. The error messages are in English and are not being translated into other languages. Some will be indecipherable by ordinary users. We judged these messages to be more useful than bare **TypeCheck** or **RangeCheck** error reports; and they have been useful during cross-checking with other JPEG implementations.

Applications are encouraged to implement a **handleerror** procedure, which routinely prints any strings in **errorinfo** and then clears the array. The default Level 2 **handleerror** procedure adequately reports these errors.

When several filters are used in cascade, it is sometimes difficult to figure out which caused a particular **ioerror**. If a DCT filter raised it, it will report something in **errorinfo**. If, for example, an **ASCII85Decode** filter (feeding a **DCTDecode** filter) raised it, then the **DCTDecode** filter would not create any **errorinfo** message. In this case, the **ioerror** would probably not have any supplementary **errorinfo** message.

11   RAM Requirements

A **DCTDecode** filter's total RAM requirement is about 7,000 bytes plus a one scanline input buffer plus whatever is required to hold one MCU-strip (see the JPEG specification), or roughly

```
7000 + (Colors * Columns) + (Columns * hs * vs * 8)/maxh bytes
```

summed over all colors, where

Colors	=	number of colors in the image
Columns	=	width of the image in pixels
maxh	=	maximum horizontal sampling value of any color
hs	=	horizontal sampling value for the color component c
vs	=	vertical sampling value for a color c

For example, on a 700 x 500 pixel RGB image with a **ColorTransform** and 2:1:1 sampling horizontally and vertically, **DCTDecode** requires about

```
7000 + (3*500) + (500*((2*2*8) + (1*1*8)+ (1*1*8))) / 2 = 20,500
bytes
```

A **DCTEncode** filter requires about 8,000 more bytes of RAM than **DCTDecode**.

For a large image, it might be important to notice that horizontal sampling reduces the RAM requirement, while vertical sampling increases it.

In a **DCTDecode** filter's typical use with the **image** operator in a printer, **image** also requires significant RAM. The combination of requirements can result in a **limitcheck** (not enough RAM) on low-RAM products. Users should not expect to print large **DCTEncode**d images on a minimum-RAM printer.

Because **DCTEncode** and **DCTDecode** use large blocks of RAM, an Insufficient RAM **limitcheck** caused by fragmentation of available RAM is possible, even though total free RAM would otherwise have sufficed. To avoid this, applications should attempt to manage RAM without fragmenting it. A freshly-booted printer might succeed after such a **limitcheck**.

12   Bugs and Incompatibilities

The following bugs are known to exist in version 2010 products. These are fixed in version 2011 products.

- **flushfile** on a **DCTDecode** filter gives an error.

- There are several **TypeCheck** versus **RangeCheck** mistakes in **DCTEncode** error reporting.

- One of the unusable **HuffTables** entry error messages prints out '0x%2X' instead of the value causing the error. Another, which is trying to print out an RST marker name prints out a negative number instead.

- An **errorinfo** message left by a **DCTDecode** or **DCTEncode** filter error, if it is not removed by the **handleerror** procedure, will lie in wait for the following error condition and be reported there instead.

- The **DCTEncode** dictionary Markers key is not interpreted in version 2010 (but the 2010 **DCTDecode** filter correctly handles COM and APPn JPEG markers).

In addition, 2011 has improved default values for **HuffTables** and **QuantTables** and an improvement in the way it handles the right and bottom edges of an image; this is discussed later. The **DCTEncode** Markers string discussed in section 19, "DCTEncode Markers String" was added for version 2011 and did not exist in 2010.

The following bugs are known to exist in version 2010 and 2011 products. These are fixed in version 2012 products:

- The *PostScript Language Reference Manual, Second Edition*, states that **ColorTransform** = 1 in the **DCTEncode** filter dictionary is ignored if **Colors** = 1 or **Colors** = 2. However, PostScript interpreters of versions 2010 and 2011 will give a **RangeCheck** error with the **errorinfo** message 'Unusable ColorTransform = 1' in this case. Version 2012 systems will correctly ignore the requested color transform.

- The *PostScript Language Reference Manual, Second Edition*, states that the scanner treats the sequence CR LF (carriage return, linefeed) as a single white space character. PostScript interpreters of versions 2010 and 2011 treat this sequence as two characters; version 2012 correctly fixes this bug. In some situations, a PostScript language program containing just one of these characters can pass through a communication channel that replaces the single CR or single LF character by the CR LF pair or characters. Clearly, if the binary DCT encoded data was subject to this transformation, it would be trashed; but sometimes the channel subtly transforms only the single CR immediately preceding the binary-encoded data; in this case PostScript interpreters of versions 2010 and 2011 will pass the LF character to the **DCTDecode** filter as its first character, and an ioerror will occur with the **errorinfo** message 'Non-baseline or invalid marker code = 10'; version 2012 systems do not have this bug.

13 Color Transforms

Adobe has implemented optional RGB-to-YCC and CMYK-to-YCCK color transforms for the **DCTEncode** filter and matching YCC-to-RGB and YCCK-to-CMYK color transforms for the **DCTDecode** filter.

The purpose of a color transform is to improve image compressibility by using a color coordinate system that separates luminance, to which the human eye is more sensitive, from chrominance which can be transmitted at lower spatial resolution. The lower spatial resolution required can be exploited during compression by downsampling and/or by choosing **QuantTables** arrays, which heavily attenuate higher frequencies.

For RGB converted to YCC with C_b and C_r downsampled twice vertically and twice horizontally, only half as many samples pass through inner loops, leading to faster execution despite the extra computation required for the color transform itself.

A disadvantage of a color transform is a small loss of dynamic range and two extra round-off errors that occur, one during encoding and the other during decoding. These errors mean that the minimum error achievable at very small quantizations is larger than if the color transform were not used. However, preliminary experiments have suggested that the quality versus compression trade-off favors use of the color transform, even for excellent quality images. Only for nearly perfect reproductions does the color transform become undesirable. Reproduction errors are discussed later.

The default of the **DCTDecode** filter is to use the YCC-to-RGB transform and to not use the YCC-to-CMYK transform. That default can be overridden by specifying a value for **ColorTransform** in the **DCTDecode** dictionary.

13.1 CMYK-to-YCCK Color Transform

The CMYK-to-YCCK color transform can only be applied to images for which the image color order is first cyan, then magenta, then yellow, and finally black; and the transformed color order is Y, C_b, C_r, and K (black).

CMYK-to-YCCK uses the same transform as RGB-to-YCC, described below, on R = (255-C), G = (255-M), and B = (255-Y), where all color sample values are integers in the range [0.255]. K is passed through unchanged.

13.2 RGB-to-YCC Color Transform

The **DCTEncode** RGB-to-YCC Color transform can only be applied to images for which the original image color order is first red, then green, then blue; and the output color order is Y, then C_b, then C_r.

The RGB-to-YCC and YCC-to-RGB transformations used are as follows:

```
Y  = .299*R + .587*G + .114*B
Cb = −.168736*R − .331264*G + .500*B + 128
Cr = .500*R − .4186876*G − .08131241*B + 128


R = Y + 1.4020*(Cr − 128)
G = Y − .3441363*(Cb − 128) − .71413636*(Cr − 128)
B = Y + 1.772*(Cb − 128)
```

The color order of these transforms is RGB and YC_bC_r. While the above numbers are believed to be compliant with the CCIR Rec. 601-1 standard, the above table contains the numbers really used in the filter, which might not be exactly the same as 601-1.

DCTEncode quantization tables specified by **QuantTables**, coding tables specified by **HuffTables**, and sampling specified by the **HSamples** and **VSamples**, apply to the color samples emerging from the color transform. For example, when the **DCTEncode** input is RGB, the RGB-to-YCC color transform is used, **HSamples** = [3 2 1], then the Y component will be sampled 3 times, C_b 2 times, and C_r 1 time. Similarly, if **QuantTables** and/or **HuffTables** are specified, then the first table applies to Y, the next to C_b, and the third to C_r.

At typical resolutions, application software presently in use has achieved good results at 2:1:1 sampling both horizontally and vertically. In other words, the good results were obtained with the luminance sampled four times (2x2) more frequently than either of the chrominance terms (1x1).

13.3 An Alternative to the DCTDecode Color Transform

YCC and YCCK should not be regarded as Level 2 color spaces; the particular color transform used is an artifact of the filter intended to increase compressibility. It is intended that the inverse transform be performed during expansion before using the image.

However, it is possible to pass an image in any color coordinate system through the **DCTDecode** filter without using the color transforms. To do this, use the **setcolorspace** operator to specify direct imaging of YCC or YCCK images using the **CIEBasedABC** color space machinery in the Level 2 interpreter. Then pass YCC or YCCK images through **DCTDecode** with no color transform; the image machinery will then perform a single composite transform from YCC or YCCK to output device coordinates.

This alternative might be faster when the color coordinates of the output device differ from those of the RGB or CMYK source image. However, the **DCTDecode** filter's color transform is likely to be faster when the image emerging from the filter's color transform is in device color coordinates.

The color transforms used by **DCTDecode** are appropriate for RGB images which comply with the CCIR Rec. 601-1 standard. Two other common RGB color standards are CCIR 709 (formerly XA/11 MOD F) and SMPTE 240M. These alternate RGB standards use slightly different equations to convert from RGB to YCC. The **CIEBasedABC** alternative to the **DCTDecode** color transform should be considered when RGB coordinates do not comply with CCIR Rec. 601-1. An example of setting up the **CIEBasedABC** color space for CCIR 709 input is given in Example 4.10 of the *PostScript Language Reference Manual, Second Edition*.

In attempting to compress a CCIR 709 or SMPTE 240M RGB image, it is possible to use the **DCTEncode** color transform based on CCIR 601. Luminance differs slightly for these three standards, so the result of compressing with the wrong transform is some luminance contamination of the chrominance components; then the spatial sensitivity of the eye to these not-quite-chrominance components will be higher than if the correct equations had been used. However, the transformed coordinates might still yield better compression than the original RGB.

14 DCTEncode HSamples, VSamples, and Blend Downsampling

As discussed in the *PostScript Language Reference Manual, Second Edition*, Level 2 implements all combinations of **HSamples** and **VSamples** values for one, two, three, and four colors. Whenever Sum(HS*VS) > 10, the sampling combination is disallowed by JPEG and will result in an error unless **Relax**=1.

The **Blend** parameter modifies **DCTEncode** downsampling as discussed in this section.

Some choices of sampling values are favored within the software implementation; choosing these will result in slightly faster execution. The particular fast and slow cases are the same for the **DCTEncode** and **DCTDecode** filters. Fast **HSamples** cases:

```
[1]
[1 1 1]
[1 1 1 1]
[2 1 1] with or without RGB-to-YCC transform
[2 1 1 2] with CMYK-to-YCCK transform.
```

In other words, un-downsampled cases, the normal color transform cases, and the 2:1:1 no-color-transform case (expected to be useful with non-RGB color coordinates) are handled by faster loops without switches and conditionals in them. **Blend**=1 slows horizontal downsampling only a little (typically, less than 4%) for both fast and slow sampling cases.

Fast **VSamples** cases:

```
[1 1 1] with or without RGB-to-YCC color transform
[2 1 1] with or without RGB-to-YCC color transform
[3 1 1] with or without RGB-to-YCC color transform
[4 1 1] with or without RGB-to-YCC color transform
[1 1 1 1] with or without CMYK-to-YCCK color transform
[2 1 1 2] with CMYK-to-YCCK color transform
[3 1 1 3] with CMYK-to-YCCK color transform
[4 1 1 4] with CMYK-to-YCCK color transform
```

These are faster only when **Blend**=0.

While JPEG allows every color component to be downsampled 1, 2, 3, or 4 times, it does not specify any particular way for the source image to be downsampled when these numbers are not all 1.

Adobe's **DCTEncode** filter implements two downsampling methods: **Chop**, if **Blend**=0 or if **Blend** does not appear in the dictionary; and **Blend**, if **Blend**=1. **Chop** should result in simple, fast downsampling and **Blend** in accurate downsampling. This feature is extensible in the sense that **Blend**=2, 3, and so on, variations can be added later. The initial implementation accepts **Blend**=1 to 65535 as legal; however, all non-zero values presently are equivalent to **Blend**=1.

Blend and **Chop** should be regarded as experimental and as hints rather than commands. **Blend** might not result in any difference in some Level 2 interpreters. Also, software and hardware solutions will do this differently; and Adobe will change its software, if experimentation reveals any advantage to doing this. The likely implementation of **Blend** and **Chop** on a Level 2 interpreter with hardware JPEG support is to ignore the **Blend** option entirely and to carry out downsampling in whatever way the hardware supports.

Table 1 shows the implementation of **Blend** and of **Chop**, where *maxh* or *maxv* is the maximum sampling value over all colors in the frame, and *hc* or *vc* is the sampling value for a particular color. The samples are numbered A, B, C, and D.

Table 1 *Blend and Chop*

maxv maxh	vc hc	Chop Blend = 0	Blend Blend = 1
2	1	A	(A+B) / 2
3	1	B	(A+2B+C) / 4
3	2	A, C	(3A+B) / 4, (B+3C) / 4
4	1	B	(A+B+C+D) / 4
4	2	A, C	(A+B)/2, (C+D) / 2
4	3	A, B, D	A, (B+C) / 2, D

The meaning of 'accurate downsampling' is pretty fuzzy. Table 1 above does something close to a linear average of samples with fudging to avoid division. The best representative value of several samples is clearly different for linear color spaces like RGB than for exponential color spaces like CIELAB. Downsampling in **Chop** mode attempts to pick a central pixel of the group being downsampled; but when there is no central pixel, then the particular sample slightly up and to the left of center is selected.

Adobe's preliminary experience on scanned images (based on little data) is that there is no visible difference in image quality between **Blend** and **Chop**. Mathematically, on one image with **HSamples** = [2 1 1] and **VSamples** = [2 1 1] using the YCC **ColorTransform**, **Blend** results in a 3% smaller average error and a 6.4% smaller mean-square error on the C_b and C_r components; it slows **DCTEncode** about 6%. Intuitively, one would expect **Blend** to be less sensitive to dropout.

15 DCTDecode Upsampling

DCTDecode upsamples the compressed image in the same way, regardless of whether or not **Blend** was used during **DCTEncode**. Table 2 shows how upsampling is done for various sampling combinations, where the 0, 1, and 2 in the Upsampling table represent the first, second, and third samples of a color that is being upsampled:

Table 2 *DCTDecode Upsampling*

maxv	*vc*	*Upsampling*	*maxh*	*hc*	*Upsampling*
2	1	0, 0	2	1	0, 0
3	1	0, 0, 0	3	1	0, 0, 0
3	2	0, 0, 1	3	2	0, (0+1) / 2, 1
4	1	0, 0, 0, 0	4	1	0, 0, 0, 0
4	2	0, 0, 1, 1	4	2	0, 0, 1, 1
4	3	0, 1, 1, 2	4	3	0, 1, 1, 2

Also, the oddball sampling cases (*maxv*=3, *vc*=2) and (*maxv*=4, *vc*=3) vertically and the same horizontally do not have pleasing solutions for either downsampling or upsampling. The loss is spread unevenly, so it is probably a bad idea to use these.

16 Default Quantization Tables and QFactor

The **DCTDecode** filter uses quantization tables supplied in the JPEG Interchange Format and is unaffected by the encoder's default values. The quantization tables in the compressed image contain one-byte unsigned integers in the range 1 to 255.

There are two 'knobs' in the **DCTEncode** filter for controlling the trade-off between compression and image quality. The simple knob is **QFactor**, which linearly scales each quantizer (discussed below) as follows: If no **QFactor** parameter is specified in the dictionary, then the default value of 1.0 is used; if **QFactor** is specified, then it must be a number in the range 0.0 to 1,000,000.0. Each unique quantization table entry is then converted to a real number, multiplied by **QFactor**, and rounded to the nearest integer in the range [1..255]. The resulting quantization table is then used to drive the **DCTEncode** filter and is transmitted in the compressed image.

For values of **QFactor** that approach 0.0, multiplication of a **QuantTables** element by **QFactor** will have a result less than 1.0 and then be raised up to 1. When all quantizers become 1, **DCTEncode** and **DCTDecode** achieve little compression but restore the original image almost exactly. As the quantizers increase, compression and image degradation increase together.

For large values of **QFactor**, the quantizers will overflow and be lowered to the maximum value of 255. At this value, little of the original image will be retained in the compressed representation. For the default **QuantTables**, the useful range of **QFactor** will be about 0.1 to 2.5.

The more complex knob on image quality versus compression is the **QuantTables** array of **Colors** quantization tables. If **QuantTables** is omitted, default arrays are used instead. Extra elements in **QuantTables** are ignored and do not cause an error.

Each of the **Colors** quantization tables can be a string of one-byte integers or an array or packed array of numbers (that is, integers or reals). The order of elements in each quantization table supplied to the Level 2 interpreter is the zigzag or snake order defined in the JPEG specification; the length of the table is 64. Study the JPEG specification to understand how to choose the quantizers, although a few comments are offered below.

To decide the number of quantization tables to include in the compressed image, the **DCTEncode** filter compares pointers to the **Colors** quantization tables; only unique arrays are transmitted with the image. Two different strings or arrays will be found to be unequal, even if all of their elements are identical. Only when a quantization table is the same string or the same array as an earlier element is the equality discovered.

The default quantization tables are not constants of Level 2 interpreters. Adobe expects to change and/or supplement these as more is learned about color spaces and image compression. The results are sensitive to resolution, viewing distance, and **QFactor**; image orientation and scaling after decompression; and to many other factors. For uniform results across Level 2 interpreters, applications using the **DCTEncode** filter must supply their own quantization tables because the Adobe defaults will vary.

16.1   Using Quantizers

The following is a brief description of how the quantizers are used. For more details, see the JPEG specification. The DCT transform converts an 8x8 block of 8-bit samples into an 8x8 block of 11-bit transform coefficients. The transform tends to concentrate the 'energy' of an image in a few of the transform coefficients, so most of the 64 coefficients will be small.

If these coefficients were coded as-is, the reverse transformation would restore the original samples almost perfectly: less than 10% of the reconstituted samples would differ from the originals and about 2-to-1 compression would typically be possible. However, before compressing, JPEG divides each coefficient by a quantizer in the range [1..255]. These divisors are taken from the **QuantTables** table corresponding to the color being compressed. After quantization, typically all but about 14 of the transform coefficients will become 0; 4 or 5 will become 1; and the rest will have larger values. This sparse matrix is then coded compactly. Decoding restores the sparse matrix and then multiplies each element by the quantizer to restore an approximation to the original transform coefficient. Finally, it performs the reverse transformation to restore an approximation to the original 8x8 block of 8-bit samples.

JPEG quantizes and codes the 8x8 coefficient block in a zigzag order starting with the upper-left DC term and ending at the lower-right term. The first (upper-left corner) term of the 8x8 coefficient block is the DC term, which measures the average value of the samples in the 8x8 block. The next few elements are visually important low-frequency AC terms, followed by successively higher-frequency terms with less visual significance.

The quantizers should be chosen to weigh each coefficient in accordance with its importance to the human visual system. This varies with the color space, the resolution of the image, the downsampling being used, and the desired quality versus compression trade-off. For good quality, the DC and low frequency AC quantizers are usually chosen to have values of about 13, with high frequency AC terms increasing to 80 to 100; for very good quality, the quantizers are halved; and for fair quality, the quantizers are doubled.

Carefully chosen quantizers take other factors into account. The original image might have excess resolution/quality, or it might be already of marginal quality. Then the DCT transform does not uniformly emphasize the coefficients, so the quantizers must be adjusted for this. Also, the human visual system is more sensitive to horizontal and vertical features than to those at odd angles, and slightly more sensitive to horizontal resolution than to vertical. Finally, the spatial sensitivity and color sensitivity of the human eye varies with the color coordinates.

17 HuffTables Specification

As with quantization tables, **DCTDecode** is driven solely by parameters included in the compressed image, so it will operate correctly regardless of the encoder's default values for **HuffTables**.

The **DCTEncode** filter uses Huffman tables supplied to it by the optional **HuffTables** entry in the filter dictionary. This entry consists of 2 x **Colors** arrays, packed arrays, or strings. **DCTEncode** is only affected by the first 2 x **Colors** tables in the **HuffTables** array; extra tables are ignored and do not cause an error. A DC table longer than 12 elements or an AC table longer than 162 elements will be rejected; these are the maximum useful lengths for these tables.

The order of the DC and AC table elements is the same as that in the JPEG specification's DHT marker segment. The first 16 one-byte elements of each array specify the number of code words of length 1 to 16, respectively. These are followed by the one-byte values in sequence. The values for DC codewords each specify the number of magnitude bits which follow; the values for AC codewords each specify a 4-bit run length of zeroes between non-zero values and 4-bit magnitude. To determine more about the format, consult the JPEG specification.

As with **QuantTables**, the encoder transmits only unique tables in the compressed image. Two code tables are found to be identical only if the pointers to them are the same. This means that a different string or array whose elements are identical to another coding table will not be found to be identical. Only when the same string or same array is used is the identity discovered.

The use of more than two different AC or DC **HuffTables** violates a JPEG Baseline limit and causes an error unless **Relax**=1 in the **DCTEncode** argument dictionary. If **Relax**=1, then the encoded image with more than two different tables will begin with an SOF1 marker indicating that it is non-Baseline. Such a compressed image will be less interchangeable than a Baseline image, although it might compress slightly better.

If the optional **HuffTables** entry does not exist in the filter's dictionary, then default arrays will be used. As with **QuantTables**, the defaults are not constants of Level 2 PostScript and will change as more is learned about image compression. In PostScript version 2010, the default tables are not very good. The defaults are better in 2011, but applications can achieve somewhat better compression with custom tables.

The **DCTEncode** filter will assume that **HuffTables** have been setup for statistics determined at **QFactor** = 1.0. When **QFactor** is some different value, it might (in some future PostScript interpreter version) modify the **HuffTables** in some (unspecified) way to increase compression. For this rea-

son, to achieve consistent encoder operation across all Level 2 products, applications should provide custom arrays for both **QuantTables** and **HuffTables** and should specify **QFactor** = 1.0 to neutralize any scaling by the **DCTEncode** filter.

The DC table elements are magnitude categories; any value outside the range [0..11] will be rejected. Within each table, two special codes are allowed: 0 (denoting end-of-block) and 0xF (denoting a zero-run of length 16). Excepting these special codes, each AC entry consists of a 4-bit zero-run length and a 4-bit magnitude category. Any entry with magnitude category outside the range [1..10] will be rejected.

18 Adobe Application-Specific JPEG Marker

Adobe uses the JPEG X'FFEE marker (or APPE marker) to record information at the time of compression such as whether or not the sample values were blended and which color transform was performed upon the data. The format of the marker is as follows.

- Two-byte length field (specifies 14 byte marker length)

- The text 'Adobe' as a five-character ASCII big-endian string

- Two-byte **DCTEncode**/**DCTDecode** version number (presently X'65)

- Two-byte flags0 0x8000 bit: Encoder used **Blend**=1 downsampling

- Two-byte flags1

- One-byte color transform code

DCTDecode ignores and skips any APPE marker segment that does not begin with the 'Adobe' 5-character string. The convention for flags0 and flags1 is that 0 bits are benign. 1 bits in flags0 pass information that is possibly useful but not essential for decoding. 1 bits in flags1 pass information essential for decoding. **DCTDecode** could reject a compressed image, if there are 1 bits in flags1 or color transform codes that it cannot interpret. The current implementation will reject only if the **Picky** option is non-zero.

The existing form of the APPE marker can be extended by defining new 'flags0' and 'flags1' flags or new color transform codes. Also, more bytes can be added to the marker. The current plan is to provide, through options in the **DCTDecode** dictionary, any parameter or option that can be specified in the application-specific marker, so applications should not have to mimic Adobe's marker. Any application trying to mimic should arrange to cooperate with Adobe.

19   DCTEncode Markers String

The **Markers** string in **DCTEncode**'s dictionary allows arbitrary data to
be inserted immediately after the JPEG SOI (Start of Image) marker that
commences a compressed JPEG Interchange Format image. It is intended to
be used only to insert COM (comment) and APPn (application) markers
in accordance with the JPEG specification. There is no error checking.

COM and APPn markers have the following syntax:

```
COM X'FF X'FE <2-byte length field> <arbitrary string>
APPn X'FF X'En <2-byte length field> <arbitrary string>
```

where X'En is X'E0 for an APP0 marker to X'EF for an APPF marker.

20   JFIF Marker

An important application of the **DCTEncode Markers** string is to include a
JFIF marker in a DCT encoded image; a JFIF marker is a JPEG APP0 marker
specially interpreted by many applications to specify size and color space
parameters of a compressed image and, optionally, a Thumbnail. At this writ-
ing, the most recent version of the emerging JFIF standard was 1.02.

A JFIF version 1.02 marker (which is an APP0 marker) can be inserted with
the following definition in the **DCTEncode** filter's argument dictionary:

```
/Markers <FF E0 00 10 4A 46 49 46 00 01 02 01 00 96 00 96 00 00> def
```

In this string, the bytes are interpreted as follows:

```
FF E0 == APP0 marker
00 10 == the marker length field (16, including the length field but
   not including the APP0 marker itself)
4A 46 49 46 00 == zero-terminated `JFIF' string
01 02 == JFIF version number
01 == units (0=aspect ratio only; 1=dots/inch; 2=dots/cm)
00 96 == Xdensity (150 dots/inch)
00 96 == Ydensity (150 dots/inch)
00 == Xthumbnail (horizontal pixel count)
00 == Ythumbnail (vertical pixel count)
```

Excluding the JFIF version number, the JFIF versions 1.00 and 1.01 markers
also have this format. Included in the marker (but not shown above) are
3*Xthumbnail*Ythumbnail bytes of the RGB thumbnail. JFIF follows a
common convention for application markers in which the marker purpose is
identified by a string at the beginning of the marker. When it is used, JFIF
must occur immediately after the JPEG SOI marker, before any other APP0
markers; and the extension markers must occur immediately after the JFIF
marker.

JFIF version 1.02 sets the JFIF version number to `01 02' and may then optionally specify the Thumbnail in a separate extension marker. To specify the Thumbnail separately, set Xthumbnail and Ythumbnail to 0 in the example above and use an extension JFXX marker immediately after the JFIF marker to specify the Thumbnail in one of three different representations. The possible formats for the JFXX extension marker are discussed in the *JPEG File Interchange Format* Version 1.02 specification, 27 August 1992.

21 Speed in DCT Filters

This section attempts to relate the speed of the DCT filters to particular parameters of the image and the controller or CPU.

The software **DCTDecode** filter on an image compressed to about 5% of its original size using **QFactor** = 1.0, **HSamples** = [2 1 1], and **VSamples** = [2 1 1] executes at about 1.5 to 2.0 seconds per megabyte of source image on a 25 MHz MIPs CPU workstation using the MIPS compiler with -O2 optimization. The **DCTEncode** filter takes 2.0 to 2.5 seconds per megabyte of source on the same image. These times can be used as a reference point for the discussion below.

Floating point arithmetic is used only to scale quantizers, so speed is not degraded much by the lack of floating point hardware in a controller. Many integer multiplications and divisions are performed; the speed of these operations is important. The current implementation is tuned for 32-bit word size CPUs, so there is some performance penalty on low-end 16-bit controllers.

Often, the **DCTDecode** filter's execution time will be swamped by a larger downstream execution time for halftoning. This depends on the relative resolutions of the source image and the device, so it is more often true on high-resolution output devices such as typesetters. Hence, **DCTDecode** performance is not of primary interest except when the output device is of lower resolution, requires no halftoning, or has hardware support for it.

On interesting images, the non-inner-loop execution time of the filter is insignificant because it can be amortized over many samples, and the filter's overall performance is determined by the inner loops. So in discussing execution speed, it suffices to discuss the following inner loops:

Table 3 *Inner loops*

Encoder	Decoder
color transform	decoding (7 code words/block typical)
strip handling	dequantization
DCT	reverse DCT
quantization (64/block)	strip handling
coding	color transform

As quantization is reduced to produce a more accurate image, time coding or decoding and dequantizing increases significantly. Downsampling directly reduces the time spent in DCT, quantization, and coding or in decoding, dequantization, or reverse DCT. Hence, applications should consider aggressively downsampling, while using smaller quantizers to recover some lost image quality.

At normal quantization, where the image is visibly just barely degraded from the original, the 8x8 blocks are compressed about 20-to-1, and 10 Huffman code words/block is typical. In this situation, software **DCTDecode** is about twice as fast as software **DCTEncode**. In an image with no quantization (that is, the quantization tables consist of elements with the value 1), **DCTEncode** will slow to about 50% of its normal speed and **DCTDecode** to about 25% of its normal speed. This slowdown is due to the huge increase in Huffman code processing from 10 code words/block to 64 code words/block.

22 Accuracy of JPEG Implementation

In testing the fast DCT used in the implementation against an accurate DCT computed slowly with double precision floating point, the following results were obtained on a small image:

```
99% of transform coefficients were identical to the accurate ones.
No transform coefficients were in error by more than +/-1.
```

In this experiment, both the accurate and fast DCTs were rounded to nearest before comparison. In the complete fast implementation, rounding does not occur at the end of the transform. Instead, extra precision is carried into the quantization step and rounding occurs after quantization.

Similarly, comparison of the implementation's reverse DCT against an accurate reverse DCT computed with double precision floating arithmetic yielded the following results on a small test image:

```
99% of regenerated characters were identical to the accurate ones.
No regenerated characters were in error by more than +/-1.
```

All tests described above apply only to the transform and reverse transform components of the system. Color transform, downsampling, and quantization cause additional losses.

Excellent results can be obtained with relatively inaccurate JPEG implementations at normal quantizations, where quantizers are all greater than 7, and coefficient errors of +/–3 have little effect. However, because the Level 2 software implementation is very accurate, the DCT filters can be used in a near-lossless mode impossible for inaccurate implementations.

23 Accuracy of Image Reproduction

To determine accuracy limits, the following experiment was performed. The quantization table was set to all 1's (that is, no loss due to quantization) by choosing **QFactor** = 0.0, **ColorTransform** was turned off, and a 1.2 megabyte image was run through the encoder and the decoder. Then color samples in the decoded image were compared against the original with the following results:

```
91.4% of color samples were unchanged.
No color samples had errors exceeding +/-1.
```

Although no errors larger than +/–1 were observed on this image, larger errors would occur in a larger study. (A crude limit on the maximum error in any sample can be estimated as follows: If every transform coefficient for a particular 8x8 block was off by 0.5 in such a way that all rounding errors added for a particular sample, then the error in that sample would be about 0.5*(1/8)*64 or +/– 4.)

The same experiment was performed on a large RGB image using the RGB-to-YCC color transform. In this case the error, larger because of the color transform, was as follows:

```
R max. err +4/-3 avg. err 0.478
G max. err +3/-3 avg. err 0.317
B max. err +4/-4 avg. err 0.587
```

A loss of dynamic range during the color transformations and two extra 8-bit rounding steps causes the increased error. The loss of dynamic range is evidenced by the sum of coefficient magnitudes for Y, C_b, and C_r in the R, G, and B equations earlier (2.402, 2.059, and 2.772, respectively) being greater than 1.

At the above accuracy levels, little compression is achieved by the JPEG method. A more interesting question is the reproduction error at larger quantization where significant compression is achieved. The empirical result of comparing reconstituted against original image samples in a large image using version 2010 to 2012 default quantizers is as follows. In rapidly changing image areas, about 8% of sample errors exceed Qmax/8 and about 1% exceed Qmax/4, where Qmax is the largest quantizer in the quantization table. In slowly changing areas, errors are determined by quantization in the upper left quadrant of the 8x8 block because coefficients elsewhere are quantized to 0 from initial magnitudes much smaller than Q[i,j]/2. The maximum sample error in a large image is typically between 0.4*Qmax and 0.8*Qmax, where Qmax is about 100 for **QFactor** = 1.0.

Total error would include an additional highly-variable component due to downsampling and a fairly predictable component due to the color transform. The color transform typically increases the maximum and average errors by 20% at normal values of **QFactor**. However, the compressed file size is also reduced by the color transform. When all things are considered, the empirical result is that, for **QFactor** greater than about 0.1 or 0.2, the color transform seems to improve compressibility, even when there is no downsampling, for any particular level of image quality.

24 Reproduction Cyclic Stability After Initial Loss

For some image editing applications, there is a worry that JPEG will intro-

maximum and average errors at stabilization are about 25% larger than the one-time errors. Total stability was reached at the fifteenth cycle without the color transform and at the twenty-third cycle with the color transform. (The **QFactor** = 0.1 color transform case stabilized down to several unstable 8x8 blocks, which it then massaged for another 8 cycles before finally reaching complete stability.)

These empirical results, which were obtained with the version 2011 implementation, might not be obtained on other images. Also, other JPEG implementations might not have as good stability properties, even though their one-time accuracy was acceptable.

The version 2010 implementation did not stabilize along the bottom and right edges of the image, where incomplete 8x8 blocks are filled out by **DCTEncode** before compressing. Adobe incorporated a superior block extension for version 2011 and later products, which has the effect of stabilizing the right and bottom edges of images. This change has no effect on one-time filter accuracy, where the version 2010 implementation is equivalent; it affects only cyclic stability.

These small experiments suggest that JPEG might be usable by applications that repeatedly compress and expand an image, if a cumulative loss 10% to 25% larger than the one-time loss is acceptable. However, a major problem will be that when an expanded image is modified and recompressed, then further losses will be introduced in all of the 8x8 sample blocks affected by the modification.

Appendix A:
PostScript Version 2011
Default Quantizers

Default quantization tables are currently as shown below. The tables are shown both in normal un-zigzagged order and in the zigzag, or snake, order defined in the JPEG specification and required for **DCTEncode**. If no colors are downsampled, then quantization Table A is used for all; if any are down-sampled, then Table A is used for colors with the maximum sampling value both horizontally and vertically, while Table B is used for downsampled colors.

Table A Luminance Table

Zigzag = PostScript language order

0x12	0x0C	0x0D	0x0D	0x0E	0x0C	0x11	0x11
0x11	0x12	0x1B	0x13	0x14	0x15	0x1B	0x22
0x1D	0x1B	0x1B	0x19	0x24	0x34	0x33	0x29
0x20	0x29	0x30	0x32	0x3F	0x40	0x3C	0x3D
0x3D	0x3C	0x43	0x45	0x50	0x51	0x49	0x47
0x49	0x4A	0x41	0x4E	0x56	0x57	0x57	0x59
0x5D	0x67	0x67	0x67	0x65	0x5B	0x67	0x67
0x67	0x67	0x67	0x67	0x67	0x67	0x67	0x67

Non-zigzag order

0x12	0x0C	0x0C	0x11	0x1B	0x22	0x32	0x3F
0x0D	0x0E	0x11	0x15	0x1D	0x30	0x40	0x41
0x0D	0x11	0x14	0x1B	0x29	0x3C	0x4A	0x4E
0x12	0x13	0x1B	0x20	0x3D	0x49	0x56	0x5B
0x1B	0x19	0x29	0x3D	0x47	0x57	0x65	0x67
0x24	0x33	0x3C	0x49	0x57	0x67	0x67	0x67
0x34	0x43	0x51	0x59	0x67	0x67	0x67	0x67
0x45	0x50	0x5D	0x67	0x67	0x67	0x67	0x67

Table B Chrominance Table

Zigzag = PostScript language order; no downsampled components

0x13	0x14	0x14	0x1A	0x17	0x1A	0x32	0x1D
0x1D	0x32	0x45	0x45	0x3B	0x45	0x45	0x67
0x67	0x57	0x57	0x67	0x67	0x67	0x67	0x67
0x67	0x67	0x67	0x67	0x67	0x67	0x67	0x67
0x67	0x67	0x67	0x67	0x67	0x67	0x67	0x67
0x67	0x67	0x67	0x67	0x67	0x67	0x67	0x67
0x67	0x67	0x67	0x67	0x67	0x67	0x67	0x67
0x67	0x67	0x67	0x67	0x67	0x67	0x67	0x67

Non-zigzag order

0x13	0x14	0x1A	0x32	0x45	0x67	0x67	0x67
0x14	0x17	0x1D	0x45	0x67	0x67	0x67	0x67
0x1A	0x1D	0x3B	0x57	0x67	0x67	0x67	0x67
0x32	0x45	0x57	0x67	0x67	0x67	0x67	0x67
0x45	0x67	0x67	0x67	0x67	0x67	0x67	0x67
0x67	0x67	0x67	0x67	0x67	0x67	0x67	0x67
0x67	0x67	0x67	0x67	0x67	0x67	0x67	0x67
0x67	0x67	0x67	0x67	0x67	0x67	0x67	0x67

Some colors downsampled; zigzag = PostScript language order

0x13	0x14	0x14	0x17	0x16	0x17	0x1A	0x1A
0x1A	0x1B	0x27	0x24	0x20	0x24	0x25	0x30
0x33	0x3B	0x3B	0x33	0x33	0x3F	0x45	0x49
0x50	0x49	0x45	0x3A	0x41	0x53	0x55	0x60
0x60	0x57	0x56	0x49	0x67	0x67	0x67	0x67
0x67	0x67	0x67	0x67	0x67	0x67	0x67	0x67
0x67	0x67	0x67	0x67	0x67	0x67	0x67	0x67
0x67	0x67	0x67	0x67	0x67	0x67	0x67	0x67

Some colors downsampled; non-zigzag order

0x13	0x14	0x17	0x1A	0x25	0x30	0x3A	0x41
0x14	0x16	0x1A	0x24	0x33	0x45	0x53	0x67
0x17	0x1A	0x20	0x3B	0x49	0x55	0x67	0x67
0x1B	0x24	0x3B	0x50	0x60	0x67	0x67	0x67
0x27	0x33	0x49	0x60	0x67	0x67	0x67	0x67
0x33	0x45	0x57	0x67	0x67	0x67	0x67	0x67
0x3F	0x56	0x67	0x67	0x67	0x67	0x67	0x67
0x49	0x67	0x67	0x67	0x67	0x67	0x67	0x67

The quantization tables are defaulted as follows when a *QuantTables* parameter is not passed in the filter's parameter dictionary: If a color transform is used, the quantization tables are assigned correctly to the luminance and chrominance components; the two chrominance tables are similar except that the one used with quantized coefficients for downsampled images quantizes high frequencies infrequently.

The particular quantization tables are not constants of Level 2 interpreters. Adobe expects to change and/or supplement these as more is learned about color spaces and image compression. An earlier version of the JPEG specification suggested Table A as a good choice for compression of the Y component and Table B as a good choice for the C_b and C_r components of the YCC color space.

Appendix B:
PostScript Version 2011
Default Huffman Tables

If the optional **HuffTables** entry does not exist in the filter's dictionary, then the default arrays appear as shown in Table A. Like the quantization tables, if no colors are downsampled, then Table A is used for all colors; if any color is downsampled, then Table A is used for those colors which have the maximum sampling value both horizontally and vertically, while Table B is used for downsampled components.

Table A (DC luminance, QFactor >= 0.25)

0x00	0x01	0x05	0x01	0x01	0x01	0x01	0x01
0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x03	0x00	0x01	0x02	0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B				

Table B (DC chrominance, QFactor >= 0.25)

0x00	0x01	0x05	0x01	0x01	0x01	0x01	0x01
0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x01	0x00	0x02	0x03	0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B				

Table A (AC luminance, QFactor >= 0.25)

0x00	0x01	0x04	0x02	0x00	0x04	0x04	0x03
0x02	0x06	0x0A	0x06	0x0B	0x04	0x0F	0x59
0x01	0x00	0x02	0x11	0x03	0x04	0x21	0x12
0x31	0x41	0x05	0x51	0x61	0x13	0x22	0x71
0x81	0x32	0x06	0x14	0x91	0xA1	0xB1	0x42
0x23	0x52	0xC1	0xD1	0x33	0x15	0x62	0x72
0x82	0x24	0x34	0x92	0x43	0x53	0xA2	0xB2
0x25	0x07	0x44	0x54	0x35	0x63	0xE1	0xF0
0xF1	0xC2	0x16	0x73	0x26	0x08	0x09	0xD2
0x0A	0x17	0x18	0x45	0x36	0x55	0x83	0x46
0x19	0x1A	0x27	0x64	0x93	0x74	0x65	0xE2
0xF2	0xA3	0xB3	0x75	0x84	0xC3	0xD3	0x56
0xE3	0xF3	0x37	0x94	0xA4	0xB4	0xC4	0xD4
0xE4	0xF4	0x85	0x95	0xA5	0xB5	0xC5	0xD5
0xE5	0xF5	0x66	0x76	0x86	0x28	0x47	0x96
0xA6	0xB6	0xC6	0xD6	0xE6	0xF6	0x29	0x57
0x67	0x38	0x39	0x2A	0x77	0x87	0x97	0xA7
0xB7	0xC7	0xD7	0xE7	0xF7	0x48	0x58	0x68
0x78	0x88	0x98	0xA8	0xB8	0xC8	0xD8	0xE8
0xF8	0x49	0x59	0x69	0x79	0x89	0x99	0xA9
0xB9	0xC9	0xD9	0xE9	0xF9	0x3A	0x4A	0x5A
0x6A	0x7A	0x8A	0x9A	0xAA	0xBA	0xCA	0xDA
0xEA	0xFA						

Table B (AC chrominance, QFactor >= 0.25)

0x00	0x01	0x04	0x00	0x05	0x01	0x05	0x03
0x06	0x07	0x0A	0x03	0x03	0x13	0x0C	0x53
0x01	0x00	0x11	0x02	0x03	0x21	0x31	0x12
0x04	0x41	0x51	0x05	0x61	0x13	0x71	0x22
0x81	0x32	0x91	0x42	0x52	0x23	0x14	0xA1
0xB1	0x33	0xC1	0xD1	0xF0	0xE1	0x62	0x72
0x82	0x92	0xF1	0x24	0x43	0x53	0x34	0x15
0xA2	0x63	0xB2	0x06	0x73	0x07	0x44	0x54
0x25	0x35	0x16	0xC2	0x26	0x08	0x09	0xD2
0x0A	0x17	0x83	0x18	0x45	0x36	0x55	0x46
0x19	0x1A	0x27	0x64	0x93	0x74	0x65	0xE2
0xF2	0xA3	0xB3	0x75	0x84	0xC3	0xD3	0x56
0xE3	0xF3	0x37	0x94	0xA4	0xB4	0xC4	0xD4
0xE4	0xF4	0x85	0x95	0xA5	0xB5	0xC5	0xD5
0xE5	0xF5	0x66	0x76	0x86	0x28	0x47	0x96
0xA6	0xB6	0xC6	0xD6	0xE6	0xF6	0x29	0x57
0x67	0x38	0x39	0x2A	0x77	0x87	0x97	0xA7
0xB7	0xC7	0xD7	0xE7	0xF7	0x48	0x58	0x68
0x78	0x88	0x98	0xA8	0xB8	0xC8	0xD8	0xE8
0xF8	0x49	0x59	0x69	0x79	0x89	0x99	0xA9
0xB9	0xC9	0xD9	0xE9	0xF9	0x3A	0x4A	0x5A
0x6A	0x7A	0x8A	0x9A	0xAA	0xBA	0xCA	0xDA
0xEA	0xFA						

Table A (DC luminance, QFactor < 0.25)

0x00	0x00	0x06	0x03	0x01	0x01	0x01	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x04	0x05	0x06	0x07	0x03	0x02	0x01	0x00
0x08	0x09	0x0A	0x0B				

Table B (DC chrominance, QFactor < 0.25)

0x00	0x00	0x06	0x03	0x01	0x01	0x01	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x05	0x04	0x06	0x07	0x03	0x02	0x01	0x08
0x00	0x09	0x0A	0x0B				

Table A (AC luminance, QFactor < 0.25)

0x00	0x01	0x03	0x02	0x04	0x03	0x05	0x04
0x04	0x0A	0x01	0x0D	0x00	0x08	0x07	0x61
0x01	0x11	0x02	0x03	0x04	0x05	0x00	0x21
0x31	0x12	0x06	0x07	0x41	0x08	0x13	0x51
0x22	0x61	0x71	0x81	0x14	0x09	0x91	0xA1
0x32	0x15	0xF0	0xB1	0x42	0x23	0xC1	0xD1
0x16	0xE1	0xF1	0x52	0x0A	0x62	0x33	0x24
0x17	0x72	0x43	0x34	0x18	0x82	0x92	0x19
0x25	0x44	0xA2	0x53	0x63	0x54	0x64	0x26
0x27	0x73	0xB2	0x83	0x93	0xA3	0x74	0x84
0x35	0x94	0xC2	0xD2	0x36	0x45	0xB3	0x46
0xA4	0xB4	0x56	0xC3	0xD3	0x55	0x28	0x1A
0xE2	0xF2	0xE3	0xF3	0xC4	0xD4	0xE4	0xF4
0x65	0x75	0x85	0x95	0xA5	0xB5	0xC5	0xD5
0xE5	0xF5	0x66	0x76	0x86	0x96	0xA6	0xB6
0xC6	0xD6	0xE6	0xF6	0x37	0x47	0x57	0x67
0x77	0x87	0x97	0xA7	0xB7	0xC7	0xD7	0xE7
0xF7	0x38	0x48	0x58	0x68	0x78	0x88	0x98
0xA8	0xB8	0xC8	0xD8	0xE8	0xF8	0x29	0x39
0x49	0x59	0x69	0x79	0x89	0x99	0xA9	0xB9
0xC9	0xD9	0xE9	0xF9	0x2A	0x3A	0x4A	0x5A
0x6A	0x7A	0x8A	0x9A	0xAA	0xBA	0xCA	0xDA
0xEA	0xFA						

Table B (AC luminance, QFactor < 0.25)

0x00	0x01	0x02	0x03	0x05	0x04	0x04	0x07
0x0B	0x07	0x04	0x07	0x0B	0x06	0x05	0x55
0x01	0x02	0x03	0x11	0x04	0x00	0x21	0x31
0x05	0x12	0x06	0x41	0x51	0x61	0x07	0x71
0x81	0x22	0x13	0x91	0x14	0x32	0xA1	0xB1
0x08	0xC1	0x42	0x52	0x23	0x15	0xD1	0x62
0x72	0x82	0x92	0xF0	0xE1	0x33	0x43	0x53
0x24	0x09	0x16	0x17	0x34	0xF1	0x25	0xA2
0x63	0x44	0xB2	0x18	0x73	0x19	0x0A	0x35
0x26	0x36	0x54	0xC2	0xD2	0x83	0x93	0x27
0x1A	0x45	0x64	0x74	0x55	0x37	0xE2	0xF2
0xA3	0xB3	0xC3	0x28	0x29	0xD3	0xE3	0xF3
0x84	0x94	0xA4	0xB4	0xC4	0xD4	0xE4	0xF4
0x65	0x75	0x85	0x95	0xA5	0xB5	0xC5	0xD5
0xE5	0xF5	0x46	0x56	0x66	0x76	0x86	0x96
0xA6	0xB6	0xC6	0xD6	0xE6	0xF6	0x47	0x57
0x67	0x77	0x87	0x97	0xA7	0xB7	0xC7	0xD7
0xE7	0xF7	0x38	0x48	0x58	0x68	0x78	0x88
0x98	0xA8	0xB8	0xC8	0xD8	0xE8	0xF8	0x39
0x49	0x59	0x69	0x79	0x89	0x99	0xA9	0xB9
0xC9	0xD9	0xE9	0xF9	0x2A	0x3A	0x4A	0x5A
0x6A	0x7A	0x8A	0x9A	0xAA	0xBA	0xCA	0xDA
0xEA	0xFA						

These tables work well with the default quantization tables. The luminance and chrominance tables are used for the indicated components when either the YCC or YCCK color space is used via a ColorTransform. The luminance table is the default for other color coordinates that have maximum sampling both horizontally and vertically. The chrominance tables are the defaults for downsampled color components.

As with quantization defaults, these defaults are not constants of Level 2 interpreters and might change over Level 2 products as more is learned about color spaces and image compression. Applications can achieve somewhat better compression for a particular level of image quality with custom tables.

Appendix C: Image Reproduction Study Results

The default quantizers in Appendix A: PostScript Version 2011 Default Quantizers are also used by Adobe Photoshop™, which scales the default matrices by a **QFactor** to achieve different quality settings. The default quantizers were chosen to have about the right relative magnitudes and to have absolute sizes that achieve 'fair' to 'good' quality at **QFactor** = 1.0.

A choice of quantizers depends heavily on excess resolution in the original image. With some excess resolution, throwing away high frequencies through downsampling or quantization wins; but if the original image has minimum resolution, then this is ineffective. There is no single choice that covers all cases effectively.

Error studies on three images using scaled default quantizers are reported below. The three images used were:

- Japan Store Front (small objects and signs seen through a glass display window in front of a store).

- Musicians (three women of different skin tones, nicely dressed with musical instruments).

- Balloons (two colorful hot air balloons against a sky background).

Japan Store Front is busy with many high frequencies (edges, text, lines, etc.). Its flaws, small ringing patterns around high frequency details, were always more visible than other flaws. To reduce these, the relative importance of luminance high frequencies was raised relative to low frequencies and to chrominance. Here are the results of mathematical error studies with Adobe Photoshop quantizer settings:

Table C.1 *For RGB (CMYK is the same):*

1)	1:1 RGB-to-YCC	QFactor = 0.00		
2)	1:1 RGB-to-YCC	QFactor = 0.05		
3)	1:1 RGB-to-YCC	QFactor = 0.10		
4)	1:1 RGB-to-YCC	QFactor = 0.20		
5)	2:1 RGB-to-YCC	QFactor = 0.25	Blend=1	
6)	2:1 RGB-to-YCC	QFactor = 0.50	Blend=1	Default
7)	2:1 RGB-to-YCC	QFactor = 1.00	Blend=1	
8)	2:1 RGB-to-YCC	QFactor = 1.60	Blend=1	
9)	2:1 RGB-to-YCC	QFactor = 2.50	Blend=1	

The exact compression results and approximate error results are as shown below. To estimate the RMS error, multiply the average error by 1.4:

Table C.2 *Japan Store Front; busy RGB image (598,752 bytes):*

	Bytes	Max. Err.	Avg. Err.
1)	452,741 (1.32:1)	3	0.6
2)	302,371 (1.98:1)	7	1.6
3)	216,978 (2.76:1)	14	2.4
4)	145,870 (4.10:1)	28	3.8
5)	91,914 (6.51:1)	34	4.1 (ignores downsampling)
6)	61,245 (9.78:1)	46	5.8 (ignores downsampling)
7)	40,857 (14.7:1)	75	7.0 (ignores downsampling)
8)	30,456 (19.7:1)	83	9.2 (ignores downsampling)
9)	22,696 (26.4:1)	108	10.2 (ignores downsampling)

Table C.3 *Musicians; average RGB image (1,997,850 bytes):*

	Bytes	Max. Err.	Avg. Err.
1)	977,339 (2.04:1)	3	0.6.
2)	520,553 (3.84:1)	7	1.3.
3)	334,520 (5.97:1)	11	1.7.
4)	221,032 (9.04:1)	23	2.3.
5)	148,951 (13.4:1)	26	2.4 (ignores downsampling)
6)	94,272 (21.2:1)	41	3.1 (ignores downsampling)
7)	61,087 (32.7:1)	61	3.5 (ignores downsampling)
8)	45,194 (44.2:1)	70	4.6 (ignores downsampling)
9)	34,092 (58.6:1)	95	5.5 (ignores downsampling)

Table C.4 *Balloons; simple RGB image (1,244,160 bytes):*

	Bytes	Max. Err.	Avg. Err.
1)	450,909 (2.76:1)	3	0.5
2)	228,660 (5.44:1)	7	1.1
3)	144,400 (8.62:1)	11	1.4
4)	94,873 (13.1:1)	17	1.7
5)	56,976 (21.8:1)	21	1.8 (ignores downsampling)
6)	35,683 (34.9:1)	34	2.1 (ignores downsampling
7)	23,769 (52.3:1)	44	2.5 (ignores downsampling)
8)	18,334 (67.9:1)	46	3.2 (ignores downsampling)
9)	14,536 (85.6:1)	54	4.0 (ignores downsampling)

Table C.5 *Balloons; simple CMYK image (1,658,880 bytes):*

	Bytes	Max. Err.	Avg. Err.
1)	650,659 (2.55:1)	3	0.5
2)	364,548 (4.55:1)	7	1.1
3)	241,071 (6.88:1)	11	1.4
4)	158,435 (10.5:1)	16	1.7
5)	110,728 (15.0:1)	20	2.1 (ignores downsampling)
6)	67,019 (24.8:1)	23	2.5 (ignores downsampling)
7)	41,425 (40.0:1)	44	2.9 (ignores downsampling)
8)	31,086 (53.4:1)	60	3.5 (ignores downsampling)
9)	24,369 (68.1:1)	75	4.3 (ignores downsampling)

In the above study, where downsampling is involved, the average error compares the original blended value to the reconstructed value after compression and expansion; in other words, it omits the error due to downsampling.

Adobe Photoshop uses the RGB-to-YCC color transform on RGB images. Most other applications use this; and our experiments have suggested that it improves the quality vs. compression tradeoff down to about **QFactor** = 0.1. Adobe Photoshop also uses the CMYK-to-YCCK color transform on CMYK images. (Color transforms increase the error at **QFactor** = 0.00 and 0.05; abandoning it would result in essentially no error at **QFactor** = 0.00 and very small error at **QFactor** = 0.05; but it seemed better to uniformly use the transform than to complicate compatibility for other applications.)

Adobe Photoshop uses the *Blend* option on all downsampled colors. This slows compression slightly while reducing the average error 15% on downsampled colors.

Adobe Photoshop switches from 2:1 sampling of chrominance to 1:1 sampling at **QFactor** = 0.2. This is probably about the right place to switch. The default quantization and Huffman coding tables also switch at this point.

Appendix D: Changes Since Earlier Versions

Changes since October 14, 1992 version

• Added the appendix, "Image Reproduction Study Results."

Changes since August 20, 1992 version

• Several references to PostScript version numbers 2010 and 2011 were changed to refer to 2010 to 2012.

• In section 13.2, "RGB-to-YCC Color Transform," the equations were modified.

• The JFIF section was broken out of section 19, "DCTEncode Markers String," and made into its own section (section 20, "JFIF Marker").

Changes since May 12, 1992 version

• The technical changes and bugs fixed in version 2012 are discussed in section 12, "Bugs and Incompatibilities."

Changes since March 31, 1992 version

• The technical changes and bugs fixed between PostScript versions 2010, 2011, and 2012 are discussed in section 12, "Bugs and Incompatibilities."

Changes since April 5, 1991 version

• This document has been completely rewritten.

Index

L

lossy compressor   5
Luminance   20
luminance   18, 19, 20, 35
LZWDecode   6
LZWEncode   6

M

magnitude categories   27

N

NoMarker   10

P

Picky   9, 11, 27

Q

QFactor   23, 33
quantization tables   23–25
quantizers
   default   35
   using   25
QuantTables   18, 24, 25

R

Relax   9, 11
Resync   10