INSTALLATION INSTRUCTIONS for the Independent JPEG Group's JPEG software

Copyright (C) 1991-2015, Thomas G. Lane, Guido Vollbeding.

This file is part of the Independent JPEG Group's software.

For conditions of distribution and use, see the accompanying README file.

This file explains how to configure and install the IJG software. We have

tried to make this software extremely portable and flexible, so that it can be

adapted to almost any environment. The downside of this decision is that the

installation process is complicated. We have provided shortcuts to simplify

the task on common systems. But in any case, you will need at least a little

familiarity with C programming and program build procedures for your system.

If you are only using this software as part of a larger program, the larger

program's installation procedure may take care of configuring the IJG code.

For example, Ghostscript's installation script will configure the IJG code.

You don't need to read this file if you just want to compile Ghostscript.

If you are on a Unix machine, you may not need to read this file at all.

Try doing

./configure

make

make test

If that doesn't complain, do

make install

(better do "make -n install" first to see if the makefile will put the files

where you want them). Read further if you run into snags or want to customize

the code for your system.

TABLE OF CONTENTS

-----------------

Before you start

Configuring the software:

using the automatic "configure" script

using one of the supplied jconfig and makefile files

by hand

Building the software

Testing the software

Installing the software

Optional stuff

Optimization

Hints for specific systems

BEFORE YOU START

================

Before installing the software you must unpack the distributed source code.

Since you are reading this file, you have probably already succeeded in this

task. However, there is a potential for error if you needed to convert the

files to the local standard text file format (for example, if you are on

MS-DOS you may have converted LF end-of-line to CR/LF). You must apply

such conversion to all the files EXCEPT those whose names begin with "test".

The test files contain binary data; if you change them in any way then the

self-test will give bad results.

Please check the last section of this file to see if there are hints for the

specific machine or compiler you are using.

CONFIGURING THE SOFTWARE

========================

To configure the IJG code for your system, you need to create two files:

\* jconfig.h: contains values for system-dependent #define symbols.

\* Makefile: controls the compilation process.

(On a non-Unix machine, you may create "project files" or some other

substitute for a Makefile. jconfig.h is needed in any environment.)

We provide three different ways to generate these files:

\* On a Unix system, you can just run the "configure" script.

\* We provide sample jconfig files and makefiles for popular machines;

if your machine matches one of the samples, just copy the right sample

files to jconfig.h and Makefile.

\* If all else fails, read the instructions below and make your own files.

Configuring the software using the automatic "configure" script

---------------------------------------------------------------

If you are on a Unix machine, you can just type

./configure

and let the configure script construct appropriate configuration files.

If you're using "csh" on an old version of System V, you might need to type

sh configure

instead to prevent csh from trying to execute configure itself.

Expect configure to run for a few minutes, particularly on slower machines;

it works by compiling a series of test programs.

Configure was created with GNU Autoconf and it follows the usual conventions

for GNU configure scripts. It makes a few assumptions that you may want to

override. You can do this by providing optional switches to configure:

\* Configure will build both static and shared libraries, if possible.

If you want to build libjpeg only as a static library, say

./configure --disable-shared

If you want to build libjpeg only as a shared library, say

./configure --disable-static

Configure uses GNU libtool to take care of system-dependent shared library

building methods.

\* Configure will use gcc (GNU C compiler) if it's available, otherwise cc.

To force a particular compiler to be selected, use the CC option, for example

./configure CC='cc'

The same method can be used to include any unusual compiler switches.

For example, on HP-UX you probably want to say

./configure CC='cc -Aa'

to get HP's compiler to run in ANSI mode.

\* The default CFLAGS setting is "-g" for non-gcc compilers, "-g -O2" for gcc.

You can override this by saying, for example,

./configure CFLAGS='-O2'

if you want to compile without debugging support.

\* Configure will set up the makefile so that "make install" will install files

into /usr/local/bin, /usr/local/man, etc. You can specify an installation

prefix other than "/usr/local" by giving configure the option "--prefix=PATH".

\* If you don't have a lot of swap space, you may need to enable the IJG

software's internal virtual memory mechanism. To do this, give the option

"--enable-maxmem=N" where N is the default maxmemory limit in megabytes.

This is discussed in more detail under "Selecting a memory manager", below.

You probably don't need to worry about this on reasonably-sized Unix machines,

unless you plan to process very large images.

Configure has some other features that are useful if you are cross-compiling

or working in a network of multiple machine types; but if you need those

features, you probably already know how to use them.

Configuring the software using one of the supplied jconfig and makefile files

-----------------------------------------------------------------------------

If you have one of these systems, you can just use the provided configuration

files:

Makefile jconfig file System and/or compiler

makefile.manx jconfig.manx Amiga, Manx Aztec C

makefile.sas jconfig.sas Amiga, SAS C

makeproj.mac jconfig.mac Apple Macintosh, Metrowerks CodeWarrior

mak\*jpeg.st jconfig.st Atari ST/STE/TT, Pure C or Turbo C

makefile.bcc jconfig.bcc MS-DOS or OS/2, Borland C

makefile.dj jconfig.dj MS-DOS, DJGPP (Delorie's port of GNU C)

makefile.mc6 jconfig.mc6 MS-DOS, Microsoft C (16-bit only)

makefile.wat jconfig.wat MS-DOS, OS/2, or Windows NT, Watcom C

makefile.vc jconfig.vc Windows NT/9x, MS Visual C++

make\*.vc6 jconfig.vc Windows NT/9x, MS Visual C++ 6

make\*.v10 jconfig.vc Windows NT/9x, MS Visual C++ 2010 (v10)

makefile.b32 jconfig.vc Windows NT/9x, Borland C++ 32-bit (bcc32)

makefile.mms jconfig.vms Digital VMS, with MMS software

makefile.vms jconfig.vms Digital VMS, without MMS software

Copy the proper jconfig file to jconfig.h and the makefile to Makefile (or

whatever your system uses as the standard makefile name). For more info see

the appropriate system-specific hints section near the end of this file.

Configuring the software by hand

--------------------------------

First, generate a jconfig.h file. If you are moderately familiar with C,

the comments in jconfig.txt should be enough information to do this; just

copy jconfig.txt to jconfig.h and edit it appropriately. Otherwise, you may

prefer to use the ckconfig.c program. You will need to compile and execute

ckconfig.c by hand --- we hope you know at least enough to do that.

ckconfig.c may not compile the first try (in fact, the whole idea is for it

to fail if anything is going to). If you get compile errors, fix them by

editing ckconfig.c according to the directions given in ckconfig.c. Once

you get it to run, it will write a suitable jconfig.h file, and will also

print out some advice about which makefile to use.

You may also want to look at the canned jconfig files, if there is one for a

system similar to yours.

Second, select a makefile and copy it to Makefile (or whatever your system

uses as the standard makefile name). The most generic makefiles we provide

are

makefile.ansi: if your C compiler supports function prototypes

makefile.unix: if not.

(You have function prototypes if ckconfig.c put "#define HAVE\_PROTOTYPES"

in jconfig.h.) You may want to start from one of the other makefiles if

there is one for a system similar to yours.

Look over the selected Makefile and adjust options as needed. In particular

you may want to change the CC and CFLAGS definitions. For instance, if you

are using GCC, set CC=gcc. If you had to use any compiler switches to get

ckconfig.c to work, make sure the same switches are in CFLAGS.

If you are on a system that doesn't use makefiles, you'll need to set up

project files (or whatever you do use) to compile all the source files and

link them into executable files cjpeg, djpeg, jpegtran, rdjpgcom, and wrjpgcom.

See the file lists in any of the makefiles to find out which files go into

each program. Note that the provided makefiles all make a "library" file

libjpeg first, but you don't have to do that if you don't want to; the file

lists identify which source files are actually needed for compression,

decompression, or both. As a last resort, you can make a batch script that

just compiles everything and links it all together; makefile.vms is an example

of this (it's for VMS systems that have no make-like utility).

Here are comments about some specific configuration decisions you'll

need to make:

Command line style

------------------

These programs can use a Unix-like command line style which supports

redirection and piping, like this:

cjpeg inputfile >outputfile

cjpeg <inputfile >outputfile

source program | cjpeg >outputfile

The simpler "two file" command line style is just

cjpeg inputfile outputfile

You may prefer the two-file style, particularly if you don't have pipes.

You MUST use two-file style on any system that doesn't cope well with binary

data fed through stdin/stdout; this is true for some MS-DOS compilers, for

example. If you're not on a Unix system, it's safest to assume you need

two-file style. (But if your compiler provides either the Posix-standard

fdopen() library routine or a Microsoft-compatible setmode() routine, you

can safely use the Unix command line style, by defining USE\_FDOPEN or

USE\_SETMODE respectively.)

To use the two-file style, make jconfig.h say "#define TWO\_FILE\_COMMANDLINE".

Selecting a memory manager

--------------------------

The IJG code is capable of working on images that are too big to fit in main

memory; data is swapped out to temporary files as necessary. However, the

code to do this is rather system-dependent. We provide five different

memory managers:

\* jmemansi.c This version uses the ANSI-standard library routine tmpfile(),

which not all non-ANSI systems have. On some systems

tmpfile() may put the temporary file in a non-optimal

location; if you don't like what it does, use jmemname.c.

\* jmemname.c This version creates named temporary files. For anything

except a Unix machine, you'll need to configure the

select\_file\_name() routine appropriately; see the comments

near the head of jmemname.c. If you use this version, define

NEED\_SIGNAL\_CATCHER in jconfig.h to make sure the temp files

are removed if the program is aborted.

\* jmemnobs.c (That stands for No Backing Store :-).) This will compile on

almost any system, but it assumes you have enough main memory

or virtual memory to hold the biggest images you work with.

\* jmemdos.c This should be used with most 16-bit MS-DOS compilers.

See the system-specific notes about MS-DOS for more info.

IMPORTANT: if you use this, define USE\_MSDOS\_MEMMGR in

jconfig.h, and include the assembly file jmemdosa.asm in the

programs. The supplied makefiles and jconfig files for

16-bit MS-DOS compilers already do both.

\* jmemmac.c Custom version for Apple Macintosh; see the system-specific

notes for Macintosh for more info.

To use a particular memory manager, change the SYSDEPMEM variable in your

makefile to equal the corresponding object file name (for example, jmemansi.o

or jmemansi.obj for jmemansi.c).

If you have plenty of (real or virtual) main memory, just use jmemnobs.c.

"Plenty" means about ten bytes for every pixel in the largest images

you plan to process, so a lot of systems don't meet this criterion.

If yours doesn't, try jmemansi.c first. If that doesn't compile, you'll have

to use jmemname.c; be sure to adjust select\_file\_name() for local conditions.

You may also need to change unlink() to remove() in close\_backing\_store().

Except with jmemnobs.c or jmemmac.c, you need to adjust the DEFAULT\_MAX\_MEM

setting to a reasonable value for your system (either by adding a #define for

DEFAULT\_MAX\_MEM to jconfig.h, or by adding a -D switch to the Makefile).

This value limits the amount of data space the program will attempt to

allocate. Code and static data space isn't counted, so the actual memory

needs for cjpeg or djpeg are typically 100 to 150Kb more than the max-memory

setting. Larger max-memory settings reduce the amount of I/O needed to

process a large image, but too large a value can result in "insufficient

memory" failures. On most Unix machines (and other systems with virtual

memory), just set DEFAULT\_MAX\_MEM to several million and forget it. At the

other end of the spectrum, for MS-DOS machines you probably can't go much

above 300K to 400K. (On MS-DOS the value refers to conventional memory only.

Extended/expanded memory is handled separately by jmemdos.c.)

BUILDING THE SOFTWARE

=====================

Now you should be able to compile the software. Just say "make" (or

whatever's necessary to start the compilation). Have a cup of coffee.

Here are some things that could go wrong:

If your compiler complains about undefined structures, you should be able to

shut it up by putting "#define INCOMPLETE\_TYPES\_BROKEN" in jconfig.h.

If you have trouble with missing system include files or inclusion of the

wrong ones, read jinclude.h. This shouldn't happen if you used configure

or ckconfig.c to set up jconfig.h.

There are a fair number of routines that do not use all of their parameters;

some compilers will issue warnings about this, which you can ignore. There

are also a few configuration checks that may give "unreachable code" warnings.

Any other warning deserves investigation.

If you don't have a getenv() library routine, define NO\_GETENV.

Also see the system-specific hints, below.

TESTING THE SOFTWARE

====================

As a quick test of functionality we've included a small sample image in

several forms:

testorig.jpg Starting point for the djpeg tests.

testimg.ppm The output of djpeg testorig.jpg

testimg.bmp The output of djpeg -bmp -colors 256 testorig.jpg

testimg.jpg The output of cjpeg testimg.ppm

testprog.jpg Progressive-mode equivalent of testorig.jpg.

testimgp.jpg The output of cjpeg -progressive -optimize testimg.ppm

(The first- and second-generation .jpg files aren't identical since the

default compression parameters are lossy.) If you can generate duplicates

of the testimg\* files then you probably have working programs.

With most of the makefiles, "make test" will perform the necessary

comparisons.

If you're using a makefile that doesn't provide the test option, run djpeg

and cjpeg by hand and compare the output files to testimg\* with whatever

binary file comparison tool you have. The files should be bit-for-bit

identical.

If the programs complain "MAX\_ALLOC\_CHUNK is wrong, please fix", then you

need to reduce MAX\_ALLOC\_CHUNK to a value that fits in type size\_t.

Try adding "#define MAX\_ALLOC\_CHUNK 65520L" to jconfig.h. A less likely

configuration error is "ALIGN\_TYPE is wrong, please fix": defining ALIGN\_TYPE

as long should take care of that one.

If the cjpeg test run fails with "Missing Huffman code table entry", it's a

good bet that you needed to define RIGHT\_SHIFT\_IS\_UNSIGNED. Go back to the

configuration step and run ckconfig.c. (This is a good plan for any other

test failure, too.)

If you are using Unix (one-file) command line style on a non-Unix system,

it's a good idea to check that binary I/O through stdin/stdout actually

works. You should get the same results from "djpeg <testorig.jpg >out.ppm"

as from "djpeg -outfile out.ppm testorig.jpg". Note that the makefiles all

use the latter style and therefore do not exercise stdin/stdout! If this

check fails, try recompiling with USE\_SETMODE or USE\_FDOPEN defined.

If it still doesn't work, better use two-file style.

If you chose a memory manager other than jmemnobs.c, you should test that

temporary-file usage works. Try "djpeg -bmp -colors 256 -max 0 testorig.jpg"

and make sure its output matches testimg.bmp. If you have any really large

images handy, try compressing them with -optimize and/or decompressing with

-colors 256 to make sure your DEFAULT\_MAX\_MEM setting is not too large.

NOTE: this is far from an exhaustive test of the JPEG software; some modules,

such as 1-pass color quantization, are not exercised at all. It's just a

quick test to give you some confidence that you haven't missed something

major.

INSTALLING THE SOFTWARE

=======================

Once you're done with the above steps, you can install the software by

copying the executable files (cjpeg, djpeg, jpegtran, rdjpgcom, and wrjpgcom)

to wherever you normally install programs. On Unix systems, you'll also want

to put the man pages (cjpeg.1, djpeg.1, jpegtran.1, rdjpgcom.1, wrjpgcom.1)

in the man-page directory. The pre-fab makefiles don't support this step

since there's such a wide variety of installation procedures on different

systems.

If you generated a Makefile with the "configure" script, you can just say

make install

to install the programs and their man pages into the standard places.

(You'll probably need to be root to do this.) We recommend first saying

make -n install

to see where configure thought the files should go. You may need to edit

the Makefile, particularly if your system's conventions for man page

filenames don't match what configure expects.

If you want to install the IJG library itself, for use in compiling other

programs besides ours, then you need to put the four include files

jpeglib.h jerror.h jconfig.h jmorecfg.h

into your include-file directory, and put the library file libjpeg.a

(extension may vary depending on system) wherever library files go.

If you generated a Makefile with "configure", it will do what it thinks

is the right thing if you say

make install-lib

OPTIONAL STUFF

==============

Progress monitor:

If you like, you can #define PROGRESS\_REPORT (in jconfig.h) to enable display

of percent-done progress reports. The routine provided in cdjpeg.c merely

prints percentages to stderr, but you can customize it to do something

fancier.

Utah RLE file format support:

We distribute the software with support for RLE image files (Utah Raster

Toolkit format) disabled, because the RLE support won't compile without the

Utah library. If you have URT version 3.1 or later, you can enable RLE

support as follows:

1. #define RLE\_SUPPORTED in jconfig.h.

2. Add a -I option to CFLAGS in the Makefile for the directory

containing the URT .h files (typically the "include"

subdirectory of the URT distribution).

3. Add -L... -lrle to LDLIBS in the Makefile, where ... specifies

the directory containing the URT "librle.a" file (typically the

"lib" subdirectory of the URT distribution).

Support for 9-bit to 12-bit deep pixel data:

The IJG code currently allows 8, 9, 10, 11, or 12 bits sample data precision.

(For color, this means 8 to 12 bits per channel, of course.) If you need to

work with deeper than 8-bit data, you can compile the IJG code for 9-bit to

12-bit operation.

To do so:

1. In jmorecfg.h, define BITS\_IN\_JSAMPLE as 9, 10, 11, or 12 rather than 8.

2. In jconfig.h, undefine BMP\_SUPPORTED, RLE\_SUPPORTED, and TARGA\_SUPPORTED,

because the code for those formats doesn't handle deeper than 8-bit data

and won't even compile. (The PPM code does work, as explained below.

The GIF code works too; it scales 8-bit GIF data to and from 12-bit

depth automatically.)

3. Compile. Don't expect "make test" to pass, since the supplied test

files are for 8-bit data.

Currently, 9-bit to 12-bit support does not work on 16-bit-int machines.

Run-time selection and conversion of data precision are currently not

supported and may be added later.

Exception: The transcoding part (jpegtran) supports all settings in a

single instance, since it operates on the level of DCT coefficients and

not sample values.

The PPM reader (rdppm.c) can read deeper than 8-bit data from either

text-format or binary-format PPM and PGM files. Binary-format PPM/PGM files

which have a maxval greater than 255 are assumed to use 2 bytes per sample,

MSB first (big-endian order). As of early 1995, 2-byte binary format is not

officially supported by the PBMPLUS library, but it is expected that a

future release of PBMPLUS will support it. Note that the PPM reader will

read files of any maxval regardless of the BITS\_IN\_JSAMPLE setting; incoming

data is automatically rescaled to maxval=MAXJSAMPLE as appropriate for the

cjpeg bit depth.

The PPM writer (wrppm.c) will normally write 2-byte binary PPM or PGM

format, maxval=MAXJSAMPLE, when compiled with BITS\_IN\_JSAMPLE>8. Since this

format is not yet widely supported, you can disable it by compiling wrppm.c

with PPM\_NORAWWORD defined; then the data is scaled down to 8 bits to make a

standard 1-byte/sample PPM or PGM file. (Yes, this means still another copy

of djpeg to keep around. But hopefully you won't need it for very long.

Poskanzer's supposed to get that new PBMPLUS release out Real Soon Now.)

Of course, if you are working with 9-bit to 12-bit data, you probably have

it stored in some other, nonstandard format. In that case you'll probably

want to write your own I/O modules to read and write your format.

Note:

The standard Huffman tables are only valid for 8-bit data precision. If

you selected more than 8-bit data precision, cjpeg uses arithmetic coding

by default. The Huffman encoder normally uses entropy optimization to

compute usable tables for higher precision. Otherwise, you'll have to

supply different default Huffman tables.

Removing code:

If you need to make a smaller version of the JPEG software, some optional

functions can be removed at compile time. See the xxx\_SUPPORTED #defines in

jconfig.h and jmorecfg.h. If at all possible, we recommend that you leave in

decoder support for all valid JPEG files, to ensure that you can read anyone's

output. Taking out support for image file formats that you don't use is the

most painless way to make the programs smaller. Another possibility is to

remove some of the DCT methods: in particular, the "IFAST" method may not be

enough faster than the others to be worth keeping on your machine. (If you

do remove ISLOW or IFAST, be sure to redefine JDCT\_DEFAULT or JDCT\_FASTEST

to a supported method, by adding a #define in jconfig.h.)

OPTIMIZATION

============

Unless you own a Cray, you'll probably be interested in making the JPEG

software go as fast as possible. This section covers some machine-dependent

optimizations you may want to try. We suggest that before trying any of

this, you first get the basic installation to pass the self-test step.

Repeat the self-test after any optimization to make sure that you haven't

broken anything.

The integer DCT routines perform a lot of multiplications. These

multiplications must yield 32-bit results, but none of their input values

are more than 16 bits wide. On many machines, notably the 680x0 and 80x86

CPUs, a 16x16=>32 bit multiply instruction is faster than a full 32x32=>32

bit multiply. Unfortunately there is no portable way to specify such a

multiplication in C, but some compilers can generate one when you use the

right combination of casts. See the MULTIPLYxxx macro definitions in

jdct.h. If your compiler makes "int" be 32 bits and "short" be 16 bits,

defining SHORTxSHORT\_32 is fairly likely to work. When experimenting with

alternate definitions, be sure to test not only whether the code still works

(use the self-test), but also whether it is actually faster --- on some

compilers, alternate definitions may compute the right answer, yet be slower

than the default. Timing cjpeg on a large PGM (grayscale) input file is the

best way to check this, as the DCT will be the largest fraction of the runtime

in that mode. (Note: some of the distributed compiler-specific jconfig files

already contain #define switches to select appropriate MULTIPLYxxx

definitions.)

If your machine has sufficiently fast floating point hardware, you may find

that the float DCT method is faster than the integer DCT methods, even

after tweaking the integer multiply macros. In that case you may want to

make the float DCT be the default method. (The only objection to this is

that float DCT results may vary slightly across machines.) To do that, add

"#define JDCT\_DEFAULT JDCT\_FLOAT" to jconfig.h. Even if you don't change

the default, you should redefine JDCT\_FASTEST, which is the method selected

by djpeg's -fast switch. Don't forget to update the documentation files

(usage.txt and/or cjpeg.1, djpeg.1) to agree with what you've done.

If access to "short" arrays is slow on your machine, it may be a win to

define type JCOEF as int rather than short. This will cost a good deal of

memory though, particularly in some multi-pass modes, so don't do it unless

you have memory to burn and short is REALLY slow.

If your compiler can compile function calls in-line, make sure the INLINE

macro in jmorecfg.h is defined as the keyword that marks a function

inline-able. Some compilers have a switch that tells the compiler to inline

any function it thinks is profitable (e.g., -finline-functions for gcc).

Enabling such a switch is likely to make the compiled code bigger but faster.

In general, it's worth trying the maximum optimization level of your compiler,

and experimenting with any optional optimizations such as loop unrolling.

(Unfortunately, far too many compilers have optimizer bugs ... be prepared to

back off if the code fails self-test.) If you do any experimentation along

these lines, please report the optimal settings to jpeg-info@jpegclub.org so

we can mention them in future releases. Be sure to specify your machine and

compiler version.

HINTS FOR SPECIFIC SYSTEMS

==========================

We welcome reports on changes needed for systems not mentioned here. Submit

'em to jpeg-info@jpegclub.org. Also, if configure or ckconfig.c is wrong

about how to configure the JPEG software for your system, please let us know.

Acorn RISC OS:

(Thanks to Simon Middleton for these hints on compiling with Desktop C.)

After renaming the files according to Acorn conventions, take a copy of

makefile.ansi, change all occurrences of 'libjpeg.a' to 'libjpeg.o' and

change these definitions as indicated:

CFLAGS= -throwback -IC: -Wn

LDLIBS=C:o.Stubs

SYSDEPMEM=jmemansi.o

LN=Link

AR=LibFile -c -o

Also add a new line '.c.o:; $(cc) $< $(cflags) -c -o $@'. Remove the

lines '$(RM) libjpeg.o' and '$(AR2) libjpeg.o' and the 'jconfig.h'

dependency section.

Copy jconfig.txt to jconfig.h. Edit jconfig.h to define TWO\_FILE\_COMMANDLINE

and CHAR\_IS\_UNSIGNED.

Run the makefile using !AMU not !Make. If you want to use the 'clean' and

'test' makefile entries then you will have to fiddle with the syntax a bit

and rename the test files.

Amiga:

SAS C 6.50 reportedly is too buggy to compile the IJG code properly.

A patch to update to 6.51 is available from SAS or AmiNet FTP sites.

The supplied config files are set up to use jmemname.c as the memory

manager, with temporary files being created on the device named by

"JPEGTMP:".

Atari ST/STE/TT:

Copy the project files makcjpeg.st, makdjpeg.st, maktjpeg.st, and makljpeg.st

to cjpeg.prj, djpeg.prj, jpegtran.prj, and libjpeg.prj respectively. The

project files should work as-is with Pure C. For Turbo C, change library

filenames "pc..." to "tc..." in each project file. Note that libjpeg.prj

selects jmemansi.c as the recommended memory manager. You'll probably want to

adjust the DEFAULT\_MAX\_MEM setting --- you want it to be a couple hundred K

less than your normal free memory. Put "#define DEFAULT\_MAX\_MEM nnnn" into

jconfig.h to do this.

To use the 68881/68882 coprocessor for the floating point DCT, add the

compiler option "-8" to the project files and replace pcfltlib.lib with

pc881lib.lib in cjpeg.prj and djpeg.prj. Or if you don't have a

coprocessor, you may prefer to remove the float DCT code by undefining

DCT\_FLOAT\_SUPPORTED in jmorecfg.h (since without a coprocessor, the float

code will be too slow to be useful). In that case, you can delete

pcfltlib.lib from the project files.

Note that you must make libjpeg.lib before making cjpeg.ttp, djpeg.ttp,

or jpegtran.ttp. You'll have to perform the self-test by hand.

We haven't bothered to include project files for rdjpgcom and wrjpgcom.

Those source files should just be compiled by themselves; they don't

depend on the JPEG library. You can use the default.prj project file

of the Pure C distribution to make the programs.

There is a bug in some older versions of the Turbo C library which causes the

space used by temporary files created with "tmpfile()" not to be freed after

an abnormal program exit. If you check your disk afterwards, you will find

cluster chains that are allocated but not used by a file. This should not

happen in cjpeg/djpeg/jpegtran, since we enable a signal catcher to explicitly

close temp files before exiting. But if you use the JPEG library with your

own code, be sure to supply a signal catcher, or else use a different

system-dependent memory manager.

Cray:

Should you be so fortunate as to be running JPEG on a Cray YMP, there is a

compiler bug in old versions of Cray's Standard C (prior to 3.1). If you

still have an old compiler, you'll need to insert a line reading

"#pragma novector" just before the loop

for (i = 1; i <= (int) htbl->bits[l]; i++)

huffsize[p++] = (char) l;

in fix\_huff\_tbl (in V5beta1, line 204 of jchuff.c and line 176 of jdhuff.c).

[This bug may or may not still occur with the current IJG code, but it's

probably a dead issue anyway...]

HP-UX:

If you have HP-UX 7.05 or later with the "software development" C compiler,

you should run the compiler in ANSI mode. If using the configure script,

say

./configure CC='cc -Aa'

(or -Ae if you prefer). If configuring by hand, use makefile.ansi and add

"-Aa" to the CFLAGS line in the makefile.

If you have a pre-7.05 system, or if you are using the non-ANSI C compiler

delivered with a minimum HP-UX system, then you must use makefile.unix

(and do NOT add -Aa); or just run configure without the CC option.

On HP 9000 series 800 machines, the HP C compiler is buggy in revisions prior

to A.08.07. If you get complaints about "not a typedef name", you'll have to

use makefile.unix, or run configure without the CC option.

Macintosh, generic comments:

The supplied user-interface files (cjpeg.c, djpeg.c, etc) are set up to

provide a Unix-style command line interface. You can use this interface on

the Mac by means of the ccommand() library routine provided by Metrowerks

CodeWarrior or Think C. This is only appropriate for testing the library,

however; to make a user-friendly equivalent of cjpeg/djpeg you'd really want

to develop a Mac-style user interface. There isn't a complete example

available at the moment, but there are some helpful starting points:

1. Sam Bushell's free "To JPEG" applet provides drag-and-drop conversion to

JPEG under System 7 and later. This only illustrates how to use the

compression half of the library, but it does a very nice job of that part.

The CodeWarrior source code is available from http://www.pobox.com/~jsam.

2. Jim Brunner prepared a Mac-style user interface for both compression and

decompression. Unfortunately, it hasn't been updated since IJG v4, and

the library's API has changed considerably since then. Still it may be of

some help, particularly as a guide to compiling the IJG code under Think C.

Jim's code is available from the Info-Mac archives, at sumex-aim.stanford.edu

or mirrors thereof; see file /info-mac/dev/src/jpeg-convert-c.hqx.

jmemmac.c is the recommended memory manager back end for Macintosh. It uses

NewPtr/DisposePtr instead of malloc/free, and has a Mac-specific

implementation of jpeg\_mem\_available(). It also creates temporary files that

follow Mac conventions. (That part of the code relies on System-7-or-later OS

functions. See the comments in jmemmac.c if you need to run it on System 6.)

NOTE that USE\_MAC\_MEMMGR must be defined in jconfig.h to use jmemmac.c.

You can also use jmemnobs.c, if you don't care about handling images larger

than available memory. If you use any memory manager back end other than

jmemmac.c, we recommend replacing "malloc" and "free" by "NewPtr" and

"DisposePtr", because Mac C libraries often have peculiar implementations of

malloc/free. (For instance, free() may not return the freed space to the

Mac Memory Manager. This is undesirable for the IJG code because jmemmgr.c

already clumps space requests.)

Macintosh, Metrowerks CodeWarrior:

The Unix-command-line-style interface can be used by defining USE\_CCOMMAND.

You'll also need to define TWO\_FILE\_COMMANDLINE to avoid stdin/stdout.

This means that when using the cjpeg/djpeg programs, you'll have to type the

input and output file names in the "Arguments" text-edit box, rather than

using the file radio buttons. (Perhaps USE\_FDOPEN or USE\_SETMODE would

eliminate the problem, but I haven't heard from anyone who's tried it.)

On 680x0 Macs, Metrowerks defines type "double" as a 10-byte IEEE extended

float. jmemmgr.c won't like this: it wants sizeof(ALIGN\_TYPE) to be a power

of 2. Add "#define ALIGN\_TYPE long" to jconfig.h to eliminate the complaint.

The supplied configuration file jconfig.mac can be used for your jconfig.h;

it includes all the recommended symbol definitions. If you have AppleScript

installed, you can run the supplied script makeproj.mac to create CodeWarrior

project files for the library and the testbed applications, then build the

library and applications. (Thanks to Dan Sears and Don Agro for this nifty

hack, which saves us from trying to maintain CodeWarrior project files as part

of the IJG distribution...)

Macintosh, Think C:

The documentation in Jim Brunner's "JPEG Convert" source code (see above)

includes detailed build instructions for Think C; it's probably somewhat

out of date for the current release, but may be helpful.

If you want to build the minimal command line version, proceed as follows.

You'll have to prepare project files for the programs; we don't include any

in the distribution since they are not text files. Use the file lists in

any of the supplied makefiles as a guide. Also add the ANSI and Unix C

libraries in a separate segment. You may need to divide the JPEG files into

more than one segment; we recommend dividing compression and decompression

modules. Define USE\_CCOMMAND in jconfig.h so that the ccommand() routine is

called. You must also define TWO\_FILE\_COMMANDLINE because stdin/stdout

don't handle binary data correctly.

On 680x0 Macs, Think C defines type "double" as a 12-byte IEEE extended float.

jmemmgr.c won't like this: it wants sizeof(ALIGN\_TYPE) to be a power of 2.

Add "#define ALIGN\_TYPE long" to jconfig.h to eliminate the complaint.

jconfig.mac should work as a jconfig.h configuration file for Think C,

but the makeproj.mac AppleScript script is specific to CodeWarrior. Sorry.

MIPS R3000:

MIPS's cc version 1.31 has a rather nasty optimization bug. Don't use -O

if you have that compiler version. (Use "cc -V" to check the version.)

Note that the R3000 chip is found in workstations from DEC and others.

MS-DOS, generic comments for 16-bit compilers:

The IJG code is designed to work well in 80x86 "small" or "medium" memory

models (i.e., data pointers are 16 bits unless explicitly declared "far";

code pointers can be either size). You may be able to use small model to

compile cjpeg or djpeg by itself, but you will probably have to use medium

model for any larger application. This won't make much difference in

performance. You \*will\* take a noticeable performance hit if you use a

large-data memory model, and you should avoid "huge" model if at all

possible. Be sure that NEED\_FAR\_POINTERS is defined in jconfig.h if you use

a small-data memory model; be sure it is NOT defined if you use a large-data

model. (The supplied makefiles and jconfig files for Borland and Microsoft C

compile in medium model and define NEED\_FAR\_POINTERS.)

The DOS-specific memory manager, jmemdos.c, should be used if possible.

It needs some assembly-code routines which are in jmemdosa.asm; make sure

your makefile assembles that file and includes it in the library. If you

don't have a suitable assembler, you can get pre-assembled object files for

jmemdosa by FTP from ftp.uu.net:/graphics/jpeg/jdosaobj.zip. (DOS-oriented

distributions of the IJG source code often include these object files.)

When using jmemdos.c, jconfig.h must define USE\_MSDOS\_MEMMGR and must set

MAX\_ALLOC\_CHUNK to less than 64K (65520L is a typical value). If your

C library's far-heap malloc() can't allocate blocks that large, reduce

MAX\_ALLOC\_CHUNK to whatever it can handle.

If you can't use jmemdos.c for some reason --- for example, because you

don't have an assembler to assemble jmemdosa.asm --- you'll have to fall

back to jmemansi.c or jmemname.c. You'll probably still need to set

MAX\_ALLOC\_CHUNK in jconfig.h, because most DOS C libraries won't malloc()

more than 64K at a time. IMPORTANT: if you use jmemansi.c or jmemname.c,

you will have to compile in a large-data memory model in order to get the

right stdio library. Too bad.

wrjpgcom needs to be compiled in large model, because it malloc()s a 64KB

work area to hold the comment text. If your C library's malloc can't

handle that, reduce MAX\_COM\_LENGTH as necessary in wrjpgcom.c.

Most MS-DOS compilers treat stdin/stdout as text files, so you must use

two-file command line style. But if your compiler has either fdopen() or

setmode(), you can use one-file style if you like. To do this, define

USE\_SETMODE or USE\_FDOPEN so that stdin/stdout will be set to binary mode.

(USE\_SETMODE seems to work with more DOS compilers than USE\_FDOPEN.) You

should test that I/O through stdin/stdout produces the same results as I/O

to explicitly named files... the "make test" procedures in the supplied

makefiles do NOT use stdin/stdout.

MS-DOS, generic comments for 32-bit compilers:

None of the above comments about memory models apply if you are using a

32-bit flat-memory-space environment, such as DJGPP or Watcom C. (And you

should use one if you have it, as performance will be much better than

8086-compatible code!) For flat-memory-space compilers, do NOT define

NEED\_FAR\_POINTERS, and do NOT use jmemdos.c. Use jmemnobs.c if the

environment supplies adequate virtual memory, otherwise use jmemansi.c or

jmemname.c.

You'll still need to be careful about binary I/O through stdin/stdout.

See the last paragraph of the previous section.

MS-DOS, Borland C:

Be sure to convert all the source files to DOS text format (CR/LF newlines).

Although Borland C will often work OK with unmodified Unix (LF newlines)

source files, sometimes it will give bogus compile errors.

"Illegal character '#'" is the most common such error. (This is true with

Borland C 3.1, but perhaps is fixed in newer releases.)

If you want one-file command line style, just undefine TWO\_FILE\_COMMANDLINE.

jconfig.bcc already includes #define USE\_SETMODE to make this work.

(fdopen does not work correctly.)

MS-DOS, Microsoft C:

makefile.mc6 works with Microsoft C, DOS Visual C++, etc. It should only

be used if you want to build a 16-bit (small or medium memory model) program.

If you want one-file command line style, just undefine TWO\_FILE\_COMMANDLINE.

jconfig.mc6 already includes #define USE\_SETMODE to make this work.

(fdopen does not work correctly.)

Note that this makefile assumes that the working copy of itself is called

"makefile". If you want to call it something else, say "makefile.mak",

be sure to adjust the dependency line that reads "$(RFILE) : makefile".

Otherwise the make will fail because it doesn't know how to create "makefile".

Worse, some releases of Microsoft's make utilities give an incorrect error

message in this situation.

Old versions of MS C fail with an "out of macro expansion space" error

because they can't cope with the macro TRACEMS8 (defined in jerror.h).

If this happens to you, the easiest solution is to change TRACEMS8 to

expand to nothing. You'll lose the ability to dump out JPEG coefficient

tables with djpeg -debug -debug, but at least you can compile.

Original MS C 6.0 is very buggy; it compiles incorrect code unless you turn

off optimization entirely (remove -O from CFLAGS). 6.00A is better, but it

still generates bad code if you enable loop optimizations (-Ol or -Ox).

MS C 8.0 crashes when compiling jquant1.c with optimization switch /Oo ...

which is on by default. To work around this bug, compile that one file

with /Oo-.

Microsoft Windows (all versions), generic comments:

Some Windows system include files define typedef boolean as "unsigned char".

The IJG code also defines typedef boolean, but we make it an "enum" by default.

This doesn't affect the IJG programs because we don't import those Windows

include files. But if you use the JPEG library in your own program, and some

of your program's files import one definition of boolean while some import the

other, you can get all sorts of mysterious problems. A good preventive step

is to make the IJG library use "unsigned char" for boolean. To do that,

add something like this to your jconfig.h file:

/\* Define "boolean" as unsigned char, not enum, per Windows custom \*/

#ifndef \_\_RPCNDR\_H\_\_ /\* don't conflict if rpcndr.h already read \*/

typedef unsigned char boolean;

#endif

#ifndef FALSE /\* in case these macros already exist \*/

#define FALSE 0 /\* values of boolean \*/

#endif

#ifndef TRUE

#define TRUE 1

#endif

#define HAVE\_BOOLEAN /\* prevent jmorecfg.h from redefining it \*/

(This is already in jconfig.vc, by the way.)

windef.h contains the declarations

#define far

#define FAR far

Since jmorecfg.h tries to define FAR as empty, you may get a compiler

warning if you include both jpeglib.h and windef.h (which windows.h

includes). To suppress the warning, you can put "#ifndef FAR"/"#endif"

around the line "#define FAR" in jmorecfg.h.

(Something like this is already in jmorecfg.h, by the way.)

When using the library in a Windows application, you will almost certainly

want to modify or replace the error handler module jerror.c, since our

default error handler does a couple of inappropriate things:

1. it tries to write error and warning messages on stderr;

2. in event of a fatal error, it exits by calling exit().

A simple stopgap solution for problem 1 is to replace the line

fprintf(stderr, "%s\n", buffer);

(in output\_message in jerror.c) with

MessageBox(GetActiveWindow(),buffer,"JPEG Error",MB\_OK|MB\_ICONERROR);

It's highly recommended that you at least do that much, since otherwise

error messages will disappear into nowhere. (Beginning with IJG v6b, this

code is already present in jerror.c; just define USE\_WINDOWS\_MESSAGEBOX in

jconfig.h to enable it.)

The proper solution for problem 2 is to return control to your calling

application after a library error. This can be done with the setjmp/longjmp

technique discussed in libjpeg.txt and illustrated in example.c. (NOTE:

some older Windows C compilers provide versions of setjmp/longjmp that

don't actually work under Windows. You may need to use the Windows system

functions Catch and Throw instead.)

The recommended memory manager under Windows is jmemnobs.c; in other words,

let Windows do any virtual memory management needed. You should NOT use

jmemdos.c nor jmemdosa.asm under Windows.

For Windows 3.1, we recommend compiling in medium or large memory model;

for newer Windows versions, use a 32-bit flat memory model. (See the MS-DOS

sections above for more info about memory models.) In the 16-bit memory

models only, you'll need to put

#define MAX\_ALLOC\_CHUNK 65520L /\* Maximum request to malloc() \*/

into jconfig.h to limit allocation chunks to 64Kb. (Without that, you'd

have to use huge memory model, which slows things down unnecessarily.)

jmemnobs.c works without modification in large or flat memory models, but to

use medium model, you need to modify its jpeg\_get\_large and jpeg\_free\_large

routines to allocate far memory. In any case, you might like to replace

its calls to malloc and free with direct calls on Windows memory allocation

functions.

You may also want to modify jdatasrc.c and jdatadst.c to use Windows file

operations rather than fread/fwrite. This is only necessary if your C

compiler doesn't provide a competent implementation of C stdio functions.

You might want to tweak the RGB\_xxx macros in jmorecfg.h so that the library

will accept or deliver color pixels in BGR sample order, not RGB; BGR order

is usually more convenient under Windows. Note that this change will break

the sample applications cjpeg/djpeg, but the library itself works fine.

Many people want to convert the IJG library into a DLL. This is reasonably

straightforward, but watch out for the following:

1. Don't try to compile as a DLL in small or medium memory model; use

large model, or even better, 32-bit flat model. Many places in the IJG code

assume the address of a local variable is an ordinary (not FAR) pointer;

that isn't true in a medium-model DLL.

2. Microsoft C cannot pass file pointers between applications and DLLs.

(See Microsoft Knowledge Base, PSS ID Number Q50336.) So jdatasrc.c and

jdatadst.c don't work if you open a file in your application and then pass

the pointer to the DLL. One workaround is to make jdatasrc.c/jdatadst.c

part of your main application rather than part of the DLL.

3. You'll probably need to modify the macros GLOBAL() and EXTERN() to

attach suitable linkage keywords to the exported routine names. Similarly,

you'll want to modify METHODDEF() and JMETHOD() to ensure function pointers

are declared in a way that lets application routines be called back through

the function pointers. These macros are in jmorecfg.h. Typical definitions

for a 16-bit DLL are:

#define GLOBAL(type) type \_far \_pascal \_loadds \_export

#define EXTERN(type) extern type \_far \_pascal \_loadds

#define METHODDEF(type) static type \_far \_pascal

#define JMETHOD(type,methodname,arglist) \

type (\_far \_pascal \*methodname) arglist

For a 32-bit DLL you may want something like

#define GLOBAL(type) \_\_declspec(dllexport) type

#define EXTERN(type) extern \_\_declspec(dllexport) type

Although not all the GLOBAL routines are actually intended to be called by

the application, the performance cost of making them all DLL entry points is

negligible.

The unmodified IJG library presents a very C-specific application interface,

so the resulting DLL is only usable from C or C++ applications. There has

been some talk of writing wrapper code that would present a simpler interface

usable from other languages, such as Visual Basic. This is on our to-do list

but hasn't been very high priority --- any volunteers out there?

Microsoft Windows, Borland C:

The provided jconfig.bcc should work OK in a 32-bit Windows environment,

but you'll need to tweak it in a 16-bit environment (you'd need to define

NEED\_FAR\_POINTERS and MAX\_ALLOC\_CHUNK). Beware that makefile.bcc will need

alteration if you want to use it for Windows --- in particular, you should

use jmemnobs.c not jmemdos.c under Windows.

Borland C++ 4.5 fails with an internal compiler error when trying to compile

jdmerge.c in 32-bit mode. If enough people complain, perhaps Borland will fix

it. In the meantime, the simplest known workaround is to add a redundant

definition of the variable range\_limit in h2v1\_merged\_upsample(), at the head

of the block that handles odd image width (about line 268 in v6 jdmerge.c):

/\* If image width is odd, do the last output column separately \*/

if (cinfo->output\_width & 1) {

register JSAMPLE \* range\_limit = cinfo->sample\_range\_limit; /\* ADD THIS \*/

cb = GETJSAMPLE(\*inptr1);

Pretty bizarre, especially since the very similar routine h2v2\_merged\_upsample

doesn't trigger the bug.

Recent reports suggest that this bug does not occur with "bcc32a" (the

Pentium-optimized version of the compiler).

Another report from a user of Borland C 4.5 was that incorrect code (leading

to a color shift in processed images) was produced if any of the following

optimization switch combinations were used:

-Ot -Og

-Ot -Op

-Ot -Om

So try backing off on optimization if you see such a problem. (Are there

several different releases all numbered "4.5"??)

Microsoft Windows, Microsoft Visual C++:

jconfig.vc should work OK with any Microsoft compiler for a 32-bit memory

model. makefile.vc is intended for command-line use. (If you are using

the Developer Studio environment, you may prefer the DevStudio project

files; see below.)

IJG JPEG 7 adds extern "C" to jpeglib.h. This avoids the need to put

extern "C" { ... } around #include "jpeglib.h" in your C++ application.

You can also force VC++ to treat the library as C++ code by renaming

all the \*.c files to \*.cpp (and adjusting the makefile to match).

In this case you also need to define the symbol DONT\_USE\_EXTERN\_C in

the configuration to prevent jpeglib.h from using extern "C".

Microsoft Windows, Microsoft Visual C++ 6 Developer Studio:

We include makefiles that should work as project files in DevStudio 6.0 or

later. There is a library makefile that builds the IJG library as a static

Win32 library, and application makefiles that build the sample applications

as Win32 console applications. (Even if you only want the library, we

recommend building the applications so that you can run the self-test.)

To use:

1. Open the command prompt, change to the main directory and execute the

command line

NMAKE /f makefile.vc setup-vc6

This will move jconfig.vc to jconfig.h and makefiles to project files.

(Note that the renaming is critical!)

2. Open the workspace file jpeg.dsw, build the library project.

(If you are using DevStudio more recent than 6.0, you'll probably

get a message saying that the project files are being updated.)

3. Open the workspace file apps.dsw, build the application projects.

4. To perform the self-test, execute the command line

NMAKE /f makefile.vc test-build

5. Move the application .exe files from `app`\Release to an

appropriate location on your path.

Microsoft Windows, Microsoft Visual C++ 2010 Developer Studio (v10):

We include makefiles that should work as project files in Visual Studio

2010 or later. There is a library makefile that builds the IJG library

as a static Win32 library, and application makefiles that build the sample

applications as Win32 console applications. (Even if you only want the

library, we recommend building the applications so that you can run the

self-test.)

To use:

1. Open the command prompt, change to the main directory and execute the

command line

NMAKE /f makefile.vc setup-v10

This will move jconfig.vc to jconfig.h and makefiles to project files.

(Note that the renaming is critical!)

2. Open the solution file jpeg.sln, build the library project.

(If you are using Visual Studio more recent than 2010 (v10), you'll

probably get a message saying that the project files are being updated.)

3. Open the solution file apps.sln, build the application projects.

4. To perform the self-test, execute the command line

NMAKE /f makefile.vc test-build

5. Move the application .exe files from `app`\Release to an

appropriate location on your path.

Note:

There seems to be an optimization bug in the compiler which causes the

self-test to fail with the color quantization option.

We have disabled optimization for the file jquant2.c in the library

project file which causes the self-test to pass properly.

OS/2, Borland C++:

Watch out for optimization bugs in older Borland compilers; you may need

to back off the optimization switch settings. See the comments in

makefile.bcc.

SGI:

On some SGI systems, you may need to set "AR2= ar -ts" in the Makefile.

If you are using configure, you can do this by saying

./configure RANLIB='ar -ts'

This change is not needed on all SGIs. Use it only if the make fails at the

stage of linking the completed programs.

On the MIPS R4000 architecture (Indy, etc.), the compiler option "-mips2"

reportedly speeds up the float DCT method substantially, enough to make it

faster than the default int method (but still slower than the fast int

method). If you use -mips2, you may want to alter the default DCT method to

be float. To do this, put "#define JDCT\_DEFAULT JDCT\_FLOAT" in jconfig.h.

VMS:

On an Alpha/VMS system with MMS, be sure to use the "/Marco=Alpha=1"

qualifier with MMS when building the JPEG package.

VAX/VMS v5.5-1 may have problems with the test step of the build procedure

reporting differences when it compares the original and test images. If the

error points to the last block of the files, it is most likely bogus and may

be safely ignored. It seems to be because the files are Stream\_LF and

Backup/Compare has difficulty with the (presumably) null padded files.

This problem was not observed on VAX/VMS v6.1 or AXP/VMS v6.1.