a1346054 Fix whitespace History

3 contributors

1783 lines (1539 sloc) | 63.7 KB ...

```
1  //--------------------------------------------------------------------------
2  // Program to pull the information out of various types of EXIF digital
3  // camera files and show it in a reasonably consistent way
4  //
5  // Version 3.06
6  //
7  // Compiling under Windows:
8  //   Make sure you have Microsoft's compiler on the path, then run make.bat
9  //
10 // Dec 1999 - Oct 2020
11 //
12 // by Matthias Wandel   http://woodgears.ca
13 //--------------------------------------------------------------------------
14 #ifdef _WIN32
15     #include <io.h>
16 #endif
17
18 #include "jhead.h"
19
20 #include <sys/stat.h>
21
22 #define JHEAD_VERSION "3.06"
23
24 // This #define turns on features that are too very specific to
25 // how I organize my photos.  Best to ignore everything inside #ifdef MATTHIAS
26 #define MATTHIAS
27
28
29 // Bitmasks for DoModify:
```

```
#define MODIFY_ANY  1
#define READ_ANY    2
#define JPEGS_ONLY  4
#define MODIFY_JPEG 5
#define READ_JPEG   6
static int DoModify  = FALSE;


static int FilesMatched;
static int FileSequence;

static const char * CurrentFile;

static const char * progname;   // program name for error messages

//--------------------------------------------------------------------------
// Command line options flags
static int TrimExif = FALSE;        // Cut off exif beyond interesting data.
static int RenameToDate = 0;        // 1=rename, 2=rename all.
#ifdef _WIN32
static int RenameAssociatedFiles = FALSE;
#endif
static char * strftime_args = NULL; // Format for new file name.
static int Exif2FileTime  = FALSE;
       int ShowTags     = FALSE;    // Do not show raw by default.
static int Quiet        = FALSE;    // Be quiet on success (like unix programs)
       int DumpExifMap  = FALSE;
static int ShowConcise  = FALSE;
static int CreateExifSection = FALSE;
static int TrimExifTrailingZeroes = FALSE;
static char * ApplyCommand = NULL;  // Apply this command to all images.
static char * FilterModel = NULL;
static int    FilterQuality = 0;
static int    ExifOnly    = FALSE;  // Only do images with exif header
static int    ProcessOnly = -1;     // 0 for baseline, 2 for progressive only, -1 for all images.
static int    PortraitOnly = FALSE; // Only do images with portrait orientation.
static time_t ExifTimeAdjust = 0;   // Timezone adjust
static time_t ExifTimeSet = 0;      // Set exif time to a value.
static char DateSet[11];
static unsigned DateSetChars = 0;
static unsigned FileTimeToExif = FALSE;

static int DeleteComments = FALSE;
static int DeleteExif = FALSE;
static int DeleteIptc = FALSE;
static int DeleteXmp = FALSE;
static int DeleteUnknown = FALSE;
static char * ThumbSaveName = NULL; // If not NULL, use this string to make up
                                    // the filename to store the thumbnail to.
```

```
79
80  static char * ThumbInsertName = NULL; // If not NULL, use this string to make up
81                                        // the filename to retrieve the thumbnail from.
82
83  static int RegenThumbnail = FALSE;
84
85  static char * ExifXferScrFile = NULL;// Extract Exif header from this file, and
86                                        // put it into the Jpegs processed.
87
88  static int EditComment = FALSE;     // Invoke an editor for editing the comment
89  static int SuppressNonFatalErrors = FALSE; // Whether or not to pint warnings on recoverable error
90
91  static char * CommentSavefileName = NULL; // Save comment to this file.
92  static char * CommentInsertfileName = NULL; // Insert comment from this file.
93  static char * CommentInsertLiteral = NULL;  // Insert this comment (from command line)
94
95  static int AutoRotate = FALSE;
96  static int ZeroRotateTagOnly = FALSE;
97
98  static int ShowFileInfo = TRUE;     // Indicates to show standard file info
99                                      // (file name, file size, file date)
100
101
102 #ifdef MATTHIAS
103     // This #ifdef to take out less than elegant stuff for editing
104     // the comments in a JPEG.  The programs rdjpgcom and wrjpgcom
105     // included with Linux distributions do a better job.
106
107     static char * AddComment = NULL; // Add this tag.
108     static char * RemComment = NULL; // Remove this tag
109     static int AutoResize = FALSE;
110 #endif // MATTHIAS
111
112 //--------------------------------------------------------------------------
113 // Error exit handler
114 //--------------------------------------------------------------------------
115 void ErrFatal(const char * msg)
116 {
117     fprintf(stderr,"\nError : %s\n", msg);
118     if (CurrentFile) fprintf(stderr,"in file '%s'\n",CurrentFile);
119     exit(EXIT_FAILURE);
120 }
121
122 //--------------------------------------------------------------------------
123 // Report non fatal errors.  Now that microsoft.net modifies exif headers,
124 // there's corrupted ones, and there could be more in the future.
125 //--------------------------------------------------------------------------
126 void ErrNonfatal(const char * msg, int a1, int a2)
127 {
```

```
    if (SuppressNonFatalErrors) return;

    fprintf(stderr,"\nNonfatal Error : ");
    if (CurrentFile) fprintf(stderr,"'%s' ",CurrentFile);
    fprintf(stderr, msg, a1, a2);
    fprintf(stderr, "\n");
}


//--------------------------------------------------------------------------
// Invoke an editor for editing a string.
//--------------------------------------------------------------------------
static int FileEditComment(char * TempFileName, char * Comment, int CommentSize)
{
    FILE * file;
    int a;
    char QuotedPath[PATH_MAX+10];

    file = fopen(TempFileName, "w");
    if (file == NULL){
        fprintf(stderr, "Can't create file '%s'\n",TempFileName);
        ErrFatal("could not create temporary file");
    }
    fwrite(Comment, CommentSize, 1, file);

    fclose(file);

    fflush(stdout); // So logs are contiguous.

    {
        char * Editor;
        Editor = getenv("EDITOR");
        if (Editor == NULL){
#ifdef _WIN32
            Editor = "notepad";
#else
            Editor = "vi";
#endif
        }
        if (strlen(Editor) > PATH_MAX) ErrFatal("env too long");

        sprintf(QuotedPath, "%s \"%s\"",Editor, TempFileName);
        a = system(QuotedPath);
    }

    if (a != 0){
        perror("Editor failed to launch");
        exit(-1);
    }
```

```
177
178     file = fopen(TempFileName, "r");
179     if (file == NULL){
180         ErrFatal("could not open temp file for read");
181     }
182
183     // Read the file back in.
184     CommentSize = fread(Comment, 1, MAX_COMMENT_SIZE, file);
185
186     fclose(file);
187
188     unlink(TempFileName);
189
190     return CommentSize;
191 }
192
193 #ifdef MATTHIAS
194 //--------------------------------------------------------------------------
195 // Modify one of the lines in the comment field.
196 // This very specific to the photo album program stuff.
197 //--------------------------------------------------------------------------
198 static char KnownTags[][10] = {"date", "desc","scan_date","author",
199                                "keyword","videograb",
200                                "show_raw","panorama","titlepix",""};
201
202 static int ModifyDescriptComment(char * OutComment, char * SrcComment)
203 {
204     char Line[500];
205     int Len;
206     int a,i;
207     unsigned l;
208     int HasScandate = FALSE;
209     int TagExists = FALSE;
210     int Modified = FALSE;
211     Len = 0;
212
213     OutComment[0] = 0;
214
215
216     for (i=0;;i++){
217         if (SrcComment[i] == '\r' || SrcComment[i] == '\n' || SrcComment[i] == 0 || Len >= 199){
218             // Process the line.
219             if (Len > 0){
220                 Line[Len] = 0;
221                 //printf("Line: '%s'\n",Line);
222                 for (a=0;;a++){
223                     l = strlen(KnownTags[a]);
224                     if (!l){
225                         // Unknown tag.  Discard it.
```

```
                        printf("Error: Unknown tag '%s'\n", Line); // Deletes the tag.
                        Modified = TRUE;
                        break;
                    }
                    if (memcmp(Line, KnownTags[a], l) == 0){
                        if (Line[l] == ' ' || Line[l] == '=' || Line[l] == 0){
                            // Its a good tag.
                            if (Line[l] == ' ') Line[l] = '='; // Use equal sign for clarity.
                            if (a == 2) break; // Delete 'orig_path' tag.
                            if (a == 3) HasScandate = TRUE;
                            if (RemComment){
                                if (strlen(RemComment) == l){
                                    if (!memcmp(Line, RemComment, l)){
                                        Modified = TRUE;
                                        break;
                                    }
                                }
                            }
                            if (AddComment){
                                // Overwrite old comment of same tag with new one.
                                if (!memcmp(Line, AddComment, l+1)){
                                    TagExists = TRUE;
                                    strncpy(Line, AddComment, sizeof(Line));
                                    Line[sizeof(Line)-1]='\0';
                                    Modified = TRUE;
                                }
                            }
                            strncat(OutComment, Line, MAX_COMMENT_SIZE-5-strlen(OutComment));
                            strcat(OutComment, "\n");
                            break;
                        }
                    }
                }
            }
            Line[Len = 0] = 0;
            if (SrcComment[i] == 0) break;
        }else{
            Line[Len++] = SrcComment[i];
        }
    }

    if (AddComment && TagExists == FALSE){
        strncat(OutComment, AddComment, MAX_COMMENT_SIZE-5-strlen(OutComment));
        strcat(OutComment, "\n");
        Modified = TRUE;
    }

    if (!HasScandate && !ImageInfo.DateTime[0]){
        // Scan date is not in the file yet, and it doesn't have one built in.  Add it.
```

```
        char Temp[40];
        sprintf(Temp, "scan_date=%s", ctime(&ImageInfo.FileDateTime));
        strncat(OutComment, Temp, MAX_COMMENT_SIZE-5-strlen(OutComment));
        Modified = TRUE;
    }
    return Modified;
}
//--------------------------------------------------------------------------
// Automatic make smaller command stuff
//--------------------------------------------------------------------------
static int AutoResizeCmdStuff(void)
{
    static char CommandString[PATH_MAX+1];
    double scale;
    float TargetSize = 1800;

    ApplyCommand = CommandString;

    scale = TargetSize / ImageInfo.Width;
    if (scale > TargetSize / ImageInfo.Height) scale = TargetSize / ImageInfo.Height;

    if (scale > 0.8){
        if (ImageInfo.QualityGuess >= 93){
            // Re-compress at lower quality.
            sprintf(CommandString, "mogrify -quality 86 &i");
            return TRUE;
        }
        printf("not resizing %dx%x '%s'\n",ImageInfo.Height, ImageInfo.Width, ImageInfo.FileName);
        return FALSE;
    }

    if (scale < 0.4) scale = 0.4; // Don't scale down by too much.

    sprintf(CommandString, "mogrify -geometry %dx%d -quality 85 &i",(int)(ImageInfo.Width*scale+0.
                                    (int)(ImageInfo.Height*scale+0.5));
    return TRUE;
}


#endif // MATTHIAS


//--------------------------------------------------------------------------
// Escape an argument such that it is interpreted literally by the shell
// (returns the number of written characters)
//--------------------------------------------------------------------------
static int shellescape(char* to, const char* from)
{
    int i, j;
```

```
    i = j = 0;

    // Enclosing characters in double quotes preserves the literal value of
    // all characters within the quotes, with the exception of $, `, and \.
    to[j++] = '"';
    while(from[i])
    {
#ifdef _WIN32
        // Under WIN32, there isn't really anything dangerous you can do with
        // escape characters, plus windows users aren't as security paranoid.
        // Hence, no need to do fancy escaping.
        to[j++] = from[i++];
#else
        switch(from[i]) {
            case '"':
            case '$':
            case '`':
            case '\\':
                to[j++] = '\\';
                // Fallthru...
            default:
                to[j++] = from[i++];
        }
#endif
        if (j >= PATH_MAX) ErrFatal("max path exceeded");
    }
    to[j++] = '"';
    return j;
}


//--------------------------------------------------------------------------
// Apply the specified command to the JPEG file.
//--------------------------------------------------------------------------
static void DoCommand(const char * FileName, int ShowIt)
{
    int a,e;
    char ExecString[PATH_MAX*3];
    char TempName[PATH_MAX+10];
    int TempUsed = FALSE;

    e = 0;

    // Generate an unused temporary file name in the destination directory
    // (a is the number of characters to copy from FileName)
    a = strlen(FileName)-1;
    while(a > 0 && FileName[a-1] != SLASH) a--;
    memcpy(TempName, FileName, a);
    strcpy(TempName+a, "XXXXXX");
```

```
373
374     // Note: Compiler will warn about mkstemp.  but I need a filename, not a file.
375     // I could just then get the file name from what mkstemp made, and pass that
376     // to the executable, but that would make for the exact same vulnerability
377     // as mktemp - that is, that between getting the random name, and making the file
378     // some other program could snatch that exact same name!
379     // also, not all platforms support mkstemp.
380     mktemp(TempName);
381
382
383     if(!TempName[0]) {
384         ErrFatal("Cannot find available temporary file name");
385     }
386
387
388     // Build the exec string.  &i and &o in the exec string get replaced by input and output files
389     for (a=0;;a++){
390         if (ApplyCommand[a] == '&'){
391             if (ApplyCommand[a+1] == 'i'){
392                 // Input file.
393                 e += shellescape(ExecString+e, FileName);
394                 a += 1;
395                 continue;
396             }
397             if (ApplyCommand[a+1] == 'o'){
398                 // Needs an output file distinct from the input file.
399                 e += shellescape(ExecString+e, TempName);
400                 a += 1;
401                 TempUsed = TRUE;
402                 continue;
403             }
404         }
405         ExecString[e++] = ApplyCommand[a];
406         if (ApplyCommand[a] == 0) break;
407     }
408
409     if (ShowIt) printf("Cmd:%s\n",ExecString);
410
411     errno = 0;
412     a = system(ExecString);
413
414     if (a || errno){
415         // A command can however fail without errno getting set or system returning an error.
416         if (errno) perror("system");
417         ErrFatal("Problem executing specified command");
418     }
419
420     if (TempUsed){
421         // Don't delete original file until we know a new one was created by the command.
```

```
        struct stat dummy;
        if (stat(TempName, &dummy) == 0){
            struct stat buf;
            int stat_result = stat(FileName, &buf);

            unlink(FileName);
            rename(TempName, FileName);
            if (stat_result == 0){
                // set Unix access rights and time to new file
                struct utimbuf mtime;
                chmod(FileName, buf.st_mode);

                mtime.actime = buf.st_atime;
                mtime.modtime = buf.st_mtime;

                utime(FileName, &mtime);
            }
        }else{
            ErrFatal("specified command did not produce expected output file");
        }
    }
}

//--------------------------------------------------------------------------
// check if this file should be skipped based on contents.
//--------------------------------------------------------------------------
static int CheckFileSkip(void)
{
    // I sometimes add code here to only process images based on certain
    // criteria - for example, only to convert non progressive Jpegs to progressives, etc..
    if(ProcessOnly >= 0 && (ImageInfo.Process & 0x0f) != ProcessOnly){
        // ProcessOnly == 0 means skip baseline oencoded jpegs
        // ProcessOnly == 2 means skip progressive oencoded jpegs
        return TRUE;
    }

    if (FilterModel){
        // Filtering processing by camera model.
        // This feature is useful when pictures from multiple cameras are collated,
        // the its found that one of the cameras has the time set incorrectly.
        if (strstr(ImageInfo.CameraModel, FilterModel) == NULL){
            // Skip.
            return TRUE;
        }
    }
    if (FilterQuality > 0){
        //Filter by above threshold quality
        if (ImageInfo.QualityGuess < FilterQuality){
            return TRUE;
```

```
471         }
472     }
473 
474     if (ExifOnly){
475         // Filtering by EXIF only.  Skip all files that have no Exif.
476         if (FindSection(M_EXIF) == NULL){
477             return TRUE;
478         }
479     }
480 
481     if (PortraitOnly == 1){
482         if (ImageInfo.Width > ImageInfo.Height) return TRUE;
483     }
484 
485     if (PortraitOnly == -1){
486         if (ImageInfo.Width < ImageInfo.Height) return TRUE;
487     }
488 
489     return FALSE;
490 }
491 
492 //--------------------------------------------------------------------------
493 // Substitute original name for '&i' if present in specified name.
494 // This to allow specifying relative names when manipulating multiple files.
495 //--------------------------------------------------------------------------
496 static void RelativeName(char * OutFileName, const char * NamePattern, const char * OrigName)
497 {
498     // If the filename contains substring "&i", then substitute the
499     // filename for that.  This gives flexibility in terms of processing
500     // multiple files at a time.
501     char * Subst;
502     Subst = strstr(NamePattern, "&i");
503     if (Subst){
504         strncpy(OutFileName, NamePattern, Subst-NamePattern);
505         OutFileName[Subst-NamePattern] = 0;
506         strncat(OutFileName, OrigName, PATH_MAX);
507         strncat(OutFileName, Subst+2, PATH_MAX);
508     }else{
509         strncpy(OutFileName, NamePattern, PATH_MAX);
510     }
511 }
512 
513 
514 #ifdef _WIN32
515 //--------------------------------------------------------------------------
516 // Rename associated files
517 //--------------------------------------------------------------------------
518 void RenameAssociated(const char * FileName, char * NewBaseName)
519 {
```

```
520     int a;
521     int PathLen;
522     int ExtPos;
523     char FilePattern[_MAX_PATH+1];
524     char NewName[_MAX_PATH+1];
525     struct _finddata_t finddata;
526     long find_handle;
527 
528     for(ExtPos = strlen(FileName);FileName[ExtPos-1] != '.';){
529         if (--ExtPos == 0) return; // No extension!
530     }
531 
532     memcpy(FilePattern, FileName, ExtPos);
533     FilePattern[ExtPos] = '*';
534     FilePattern[ExtPos+1] = '\0';
535 
536     for(PathLen = strlen(FileName);FileName[PathLen-1] != SLASH;){
537         if (--PathLen == 0) break;
538     }
539 
540     find_handle = _findfirst(FilePattern, &finddata);
541 
542     for (;;){
543         if (find_handle == -1) break;
544 
545         // Eliminate the obvious patterns.
546         if (!memcmp(finddata.name, ".",2)) goto next_file;
547         if (!memcmp(finddata.name, "..",3)) goto next_file;
548         if (finddata.attrib & _A_SUBDIR) goto next_file;
549 
550         strncpy(FilePattern+PathLen, finddata.name, PATH_MAX-PathLen); // full name with path
551 
552         strcpy(NewName, NewBaseName);
553         for(a = strlen(finddata.name);finddata.name[a] != '.';){
554             if (--a == 0) goto next_file;
555         }
556         strncat(NewName, finddata.name+a, _MAX_PATH-strlen(NewName)); // add extension to new name
557 
558         if (rename(FilePattern, NewName) == 0){
559             if (!Quiet){
560                 printf("%s --> %s\n",FilePattern, NewName);
561             }
562         }
563 
564         next_file:
565         if (_findnext(find_handle, &finddata) != 0) break;
566     }
567     _findclose(find_handle);
568 }
```

```
#endif

//--------------------------------------------------------------------------
// Handle renaming of files by date.
//--------------------------------------------------------------------------
static void DoFileRenaming(const char * FileName)
{
    int PrefixPart = 0; // Where the actual filename starts.
    int ExtensionPart;  // Where the file extension starts.
    int a;
    struct tm tm;
    char NewBaseName[PATH_MAX*2];
    int AddLetter = 0;
    char NewName[PATH_MAX+2];

    ExtensionPart = strlen(FileName);
    for (a=0;FileName[a];a++){
        if (FileName[a] == SLASH){
            // Don't count path component.
            PrefixPart = a+1;
        }

        if (FileName[a] == '.') ExtensionPart = a;  // Remember where extension starts.
    }
    if (ExtensionPart < PrefixPart) { // no extension found
        ExtensionPart = strlen(FileName);
    }

    if (!Exif2tm(&tm, ImageInfo.DateTime)){
        printf("File '%s' contains no exif date stamp.  Using file date\n",FileName);
        // Use file date/time instead.
        tm = *localtime(&ImageInfo.FileDateTime);
    }


    strncpy(NewBaseName, FileName, PATH_MAX); // Get path component of name.

    if (strftime_args){
        // Complicated scheme for flexibility.  Just pass the args to strftime.
        time_t UnixTime;

        char *s;
        char pattern[PATH_MAX+20];
        int n = ExtensionPart - PrefixPart;

        // Call mktime to get weekday and such filled in.
        UnixTime = mktime(&tm);
        if ((int)UnixTime == -1){
            printf("Could not convert %s to unix time",ImageInfo.DateTime);
```

```
618             return;
619         }
620 
621         // Substitute "%f" for the original name (minus path & extension)
622         pattern[PATH_MAX-1]=0;
623         strncpy(pattern, strftime_args, PATH_MAX-1);
624         while ((s = strstr(pattern, "%f")) && strlen(pattern) + n < PATH_MAX-1){
625             memmove(s + n, s + 2, strlen(s+2) + 1);
626             memmove(s, FileName + PrefixPart, n);
627         }
628 
629         {
630             // Sequential number renaming part.
631             // '%i' type pattern becomes sequence number.
632             int ppos = -1;
633             for (a=0;pattern[a];a++){
634                 if (pattern[a] == '%'){
635                      ppos = a;
636                 }else if (pattern[a] == 'i'){
637                     if (ppos >= 0 && a<ppos+4){
638                         // Replace this part with a number.
639                         char pat[8], num[16];
640                         int l,nl;
641                         memcpy(pat, pattern+ppos, 4);
642                         pat[a-ppos] = 'd'; // Replace 'i' with 'd' for '%d'
643                         pat[a-ppos+1] = '\0';
644                         sprintf(num, pat, FileSequence); // let printf do the number formatting.
645                         nl = strlen(num);
646                         l = strlen(pattern+a+1);
647                         if (ppos+nl+l+1 >= PATH_MAX) ErrFatal("str overflow");
648                         memmove(pattern+ppos+nl, pattern+a+1, l+1);
649                         memcpy(pattern+ppos, num, nl);
650                         break;
651                     }
652                 }else if (!isdigit(pattern[a])){
653                     ppos = -1;
654                 }
655             }
656         }
657         strftime(NewName, PATH_MAX, pattern, &tm);
658     }else{
659         // My favourite scheme.
660         sprintf(NewName, "%02d%02d-%02d%02d%02d",
661              tm.tm_mon+1, tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec);
662     }
663 
664     NewBaseName[PrefixPart] = 0;
665     CatPath(NewBaseName, NewName);
666 
```

```
667     AddLetter = isdigit(NewBaseName[strlen(NewBaseName)-1]);
668     for (a=0;;a++){
669         char NewName[PATH_MAX*2+10];
670         char NameExtra[3];
671         struct stat dummy;
672 
673         if (a){
674             // Generate a suffix for the file name if previous choice of names is taken.
675             // depending on whether the name ends in a letter or digit, pick the opposite to separ
676             // it.  This to avoid using a separator character - this because any good separator
677             // is before the '.' in ascii, and so sorting the names would put the later name befor
678             // the name without suffix, causing the pictures to more likely be out of order.
679             if (AddLetter){
680                 NameExtra[0] = (char)('a'-1+a); // Try a,b,c,d... for suffix if it ends in a numbe
681             }else{
682                 NameExtra[0] = (char)('0'-1+a); // Try 0,1,2,3... for suffix if it ends in a latte
683             }
684             NameExtra[1] = 0;
685         }else{
686             NameExtra[0] = 0;
687         }
688 
689         snprintf(NewName, sizeof(NewName), "%s%s.jpg", NewBaseName, NameExtra);
690 
691         if (!strcmp(FileName, NewName)) break; // Skip if its already this name.
692 
693         if (!EnsurePathExists(NewBaseName)){
694             break;
695         }
696 
697 
698         if (stat(NewName, &dummy)){
699             // This name does not pre-exist.
700             if (rename(FileName, NewName) == 0){
701                 printf("%s --> %s\n",FileName, NewName);
702 #ifdef _WIN32
703                 if (RenameAssociatedFiles){
704                     sprintf(NewName, "%s%s", NewBaseName, NameExtra);
705                     RenameAssociated(FileName, NewName);
706                 }
707 #endif
708             }else{
709                 printf("Error: Couldn't rename '%s' to '%s'\n",FileName, NewName);
710             }
711             break;
712 
713         }
714 
715         if (a > 25 || (!AddLetter && a > 9)){
```

```
            printf("Possible new names for for '%s' already exist\n",FileName);
            break;
        }
    }
}

//--------------------------------------------------------------------------
// Rotate the image and its thumbnail
//--------------------------------------------------------------------------
static int DoAutoRotate(const char * FileName)
{
    if (ImageInfo.Orientation != 1){
        const char * Argument;
        Argument = ClearOrientation();
        if (Argument == NULL) return FALSE; // orientation tag in image, nothing changed.

        if (!ZeroRotateTagOnly){
            char RotateCommand[PATH_MAX*2+50];
            if (strlen(Argument) == 0){
                // Unknown orientation, but still modified.
                return TRUE; // Image is still modified.
            }
            sprintf(RotateCommand, "jpegtran -trim -%s -outfile &o &i", Argument);
            ApplyCommand = RotateCommand;
            DoCommand(FileName, FALSE);
            ApplyCommand = NULL;

            // Now rotate the thumbnail, if there is one.
            if (ImageInfo.ThumbnailOffset &&
                ImageInfo.ThumbnailSize &&
                ImageInfo.ThumbnailAtEnd){
                // Must have a thumbnail that exists and is modifiable.

                char ThumbTempName_in[PATH_MAX+5];
                char ThumbTempName_out[PATH_MAX+5];

                strcpy(ThumbTempName_in, FileName);
                strcat(ThumbTempName_in, ".thi");
                strcpy(ThumbTempName_out, FileName);
                strcat(ThumbTempName_out, ".tho");
                SaveThumbnail(ThumbTempName_in);
                sprintf(RotateCommand,"jpegtran -trim -%s -outfile \"%s\" \"%s\"",
                    Argument, ThumbTempName_out, ThumbTempName_in);

                if (system(RotateCommand) == 0){
                    // Put the thumbnail back in the header
                    ReplaceThumbnail(ThumbTempName_out);
                }

```

```
765                 unlink(ThumbTempName_in);
766                 unlink(ThumbTempName_out);
767             }
768         }
769         return TRUE;
770     }
771     return FALSE;
772 }
773 
774 //--------------------------------------------------------------------------
775 // Regenerate the thumbnail using mogrify
776 //--------------------------------------------------------------------------
777 static int RegenerateThumbnail(const char * FileName)
778 {
779     char ThumbnailGenCommand[PATH_MAX*2+50];
780     if (ImageInfo.ThumbnailOffset == 0 || ImageInfo.ThumbnailAtEnd == FALSE){
781         // There is no thumbnail, or the thumbnail is not at the end.
782         return FALSE;
783     }
784 
785     sprintf(ThumbnailGenCommand, "mogrify -thumbnail %dx%d -quality 80 \"%s\"",
786         RegenThumbnail, RegenThumbnail, FileName);
787 
788     if (system(ThumbnailGenCommand) == 0){
789         // Put the thumbnail back in the header
790         return ReplaceThumbnail(FileName);
791     }else{
792         ErrFatal("Unable to run 'mogrify' command");
793         return FALSE;
794     }
795 }
796 
797 //--------------------------------------------------------------------------
798 // Set file time as exif time.
799 //--------------------------------------------------------------------------
800 void FileTimeAsString(char * TimeStr)
801 {
802     struct tm ts;
803     ts = *localtime(&ImageInfo.FileDateTime);
804     strftime(TimeStr, 20, "%Y:%m:%d %H:%M:%S", &ts);
805 }
806 
807 //--------------------------------------------------------------------------
808 // Do selected operations to one file at a time.
809 //--------------------------------------------------------------------------
810 static void ProcessFile(const char * FileName)
811 {
812     int Modified = FALSE;
813     ReadMode_t ReadMode;
```

```
814
815     if (strlen(FileName) >= PATH_MAX-1){
816         // Protect against buffer overruns in strcpy / strcat's on filename
817         ErrFatal("filename too long");
818     }
819
820     ReadMode = READ_METADATA;
821     CurrentFile = FileName;
822     FilesMatched = 1;
823
824     ResetJpgfile();
825     Clear_EXIF();
826
827     // Start with an empty image information structure.
828     memset(&ImageInfo, 0, sizeof(ImageInfo));
829     ImageInfo.FlashUsed = -1;
830     ImageInfo.MeteringMode = -1;
831     ImageInfo.Whitebalance = -1;
832
833     // Store file date/time.
834     {
835         struct stat st;
836         if (stat(FileName, &st) >= 0){
837             ImageInfo.FileDateTime = st.st_mtime;
838             ImageInfo.FileSize = st.st_size;
839         }else{
840             ErrFatal("No such file");
841         }
842     }
843
844     if ((DoModify & MODIFY_ANY) || RenameToDate || Exif2FileTime){
845         if (access(FileName, 2 /*W_OK*/)){
846             printf("Skipping readonly file '%s'\n",FileName);
847             return;
848         }
849     }
850
851     strncpy(ImageInfo.FileName, FileName, PATH_MAX);
852
853
854     if (ApplyCommand || AutoRotate){
855         // Applying a command is special - the headers from the file have to be
856         // pre-read, then the command executed, and then the image part of the file read.
857
858         if (!ReadJpegFile(FileName, READ_METADATA)) return;
859
860         #ifdef MATTHIAS
861             if (AutoResize){
862                 // Automatic resize computation - to customize for each run...
```

```
863                 if (AutoResizeCmdStuff() == 0){
864                     DiscardData();
865                     return;
866                 }
867             }
868         #endif // MATTHIAS
869
870
871         if (CheckFileSkip()){
872             DiscardData();
873             return;
874         }
875
876         DiscardAllButExif();
877
878         if (AutoRotate){
879             if (DoAutoRotate(FileName)){
880                 Modified = TRUE;
881             }
882         }else{
883             struct stat dummy;
884             DoCommand(FileName, Quiet ? FALSE : TRUE);
885
886             if (stat(FileName, &dummy)){
887                 // The file is not there anymore. Perhaps the command
888                 // was a delete or a move.  So we are all done.
889                 return;
890             }
891             Modified = TRUE;
892         }
893         ReadMode = READ_IMAGE;   // Don't re-read exif section again on next read.
894     }
895
896     if (ExifXferScrFile){
897         char RelativeExifName[PATH_MAX+1];
898
899         // Make a relative name.
900         RelativeName(RelativeExifName, ExifXferScrFile, FileName);
901
902         if(!ReadJpegFile(RelativeExifName, READ_METADATA)) return;
903
904         DiscardAllButExif();    // Don't re-read exif section again on next read.
905
906         Modified = TRUE;
907         ReadMode = READ_IMAGE;
908     }
909
910     if (DoModify){
911         ReadMode |= READ_IMAGE;
```

```
912     }
913
914     if (!ReadJpegFile(FileName, ReadMode)) return;
915
916     if (CheckFileSkip()){
917         DiscardData();
918         return;
919     }
920
921     if (TrimExifTrailingZeroes){
922         if (ImageInfo.ThumbnailAtEnd){
923             Section_t * ExifSection;
924             int NumRedundant, WasRedundant;
925             unsigned char * StartRedundant;
926             //printf("Exif: Thumbnail %d - %d\n",ImageInfo.ThumbnailOffset, ImageInfo.ThumbnailOff
927             ExifSection = FindSection(M_EXIF);
928
929             StartRedundant = ExifSection->Data + 8 + ImageInfo.ThumbnailOffset+ImageInfo.Thumbnail
930             WasRedundant = NumRedundant = (ExifSection->Size) - (ImageInfo.ThumbnailOffset + Image
931
932             //printf("Exif length: %d  Wasted: %d\n",ExifSection->Size, NumRedundant);
933
934             for(;NumRedundant > 0 && StartRedundant[NumRedundant-1] == 0;NumRedundant--);// Only r
935
936             if (NumRedundant != WasRedundant){
937                 int NewExifSize;
938                 printf("Trimming %d bytes from exif in %s\n", WasRedundant-NumRedundant, FileName)
939                 NewExifSize = ImageInfo.ThumbnailOffset + ImageInfo.ThumbnailSize + 8 + NumRedunda
940                 ExifSection->Data[0] = (uchar)(NewExifSize >> 8); // Must write new length into ex
941                 ExifSection->Data[1] = (uchar)NewExifSize;
942                 ExifSection->Size = NewExifSize;
943                 Modified = TRUE;
944             }else{
945                 //printf("Noting to remove from %s\n", FileName);
946             }
947         }
948     }
949
950     FileSequence += 1; // Count files processed.
951
952     if (ShowConcise){
953         ShowConciseImageInfo();
954     }else{
955         if (!(DoModify) || ShowTags){
956             ShowImageInfo(ShowFileInfo);
957
958             {
959                 // if IPTC section is present, show it also.
960                 Section_t * IptcSection;
```

```
961                 IptcSection = FindSection(M_IPTC);
962
963                 if (IptcSection){
964                     show_IPTC(IptcSection->Data, IptcSection->Size);
965                 }
966             }
967             printf("\n");
968         }
969     }
970
971     if (ThumbSaveName){
972         char OutFileName[PATH_MAX+1];
973         // Make a relative name.
974         RelativeName(OutFileName, ThumbSaveName, FileName);
975
976         if (SaveThumbnail(OutFileName)){
977             printf("Created: '%s'\n", OutFileName);
978         }
979     }
980
981     if (CreateExifSection){
982         // Make a new minimal exif section
983         create_EXIF();
984         Modified = TRUE;
985     }
986
987     if (RegenThumbnail){
988         if (RegenerateThumbnail(FileName)){
989             Modified = TRUE;
990         }
991     }
992
993     if (ThumbInsertName){
994         char ThumbFileName[PATH_MAX+1];
995         // Make a relative name.
996         RelativeName(ThumbFileName, ThumbInsertName, FileName);
997
998         if (ReplaceThumbnail(ThumbFileName)){
999             Modified = TRUE;
1000        }
1001    }else if (TrimExif){
1002        // Deleting thumbnail is just replacing it with a null thumbnail.
1003        if (ReplaceThumbnail(NULL)){
1004            Modified = TRUE;
1005        }
1006    }
1007
1008    if (
1009#ifdef MATTHIAS
```

```
1010            AddComment || RemComment ||
1011    #endif
1012                        EditComment || CommentInsertfileName || CommentInsertLiteral){
1013
1014            Section_t * CommentSec;
1015            char Comment[MAX_COMMENT_SIZE+1];
1016            int CommentSize;
1017
1018            CommentSec = FindSection(M_COM);
1019
1020            if (CommentSec == NULL){
1021                unsigned char * DummyData;
1022
1023                DummyData = (uchar *) malloc(3);
1024                DummyData[0] = 0;
1025                DummyData[1] = 2;
1026                DummyData[2] = 0;
1027                CommentSec = CreateSection(M_COM, DummyData, 2);
1028            }
1029
1030            CommentSize = CommentSec->Size-2;
1031            if (CommentSize > MAX_COMMENT_SIZE){
1032                fprintf(stderr, "Truncating comment at %d chars\n",MAX_COMMENT_SIZE);
1033                CommentSize = MAX_COMMENT_SIZE;
1034            }
1035
1036            if (CommentInsertfileName){
1037                // Read a new comment section from file.
1038                char CommentFileName[PATH_MAX+1];
1039                FILE * CommentFile;
1040
1041                // Make a relative name.
1042                RelativeName(CommentFileName, CommentInsertfileName, FileName);
1043
1044                CommentFile = fopen(CommentFileName,"r");
1045                if (CommentFile == NULL){
1046                    printf("Could not open '%s'\n",CommentFileName);
1047                }else{
1048                    // Read it in.
1049                    // Replace the section.
1050                    CommentSize = fread(Comment, 1, MAX_COMMENT_SIZE, CommentFile);
1051                    fclose(CommentFile);
1052                    if (CommentSize < 0) CommentSize = 0;
1053                }
1054            }else if (CommentInsertLiteral){
1055                strncpy(Comment, CommentInsertLiteral, MAX_COMMENT_SIZE);
1056                CommentSize = strlen(Comment);
1057            }else{
1058    #ifdef MATTHIAS
```

```
1059            char CommentZt[MAX_COMMENT_SIZE+1];
1060            memcpy(CommentZt, (char *)CommentSec->Data+2, CommentSize);
1061            CommentZt[CommentSize] = '\0';
1062            if (ModifyDescriptComment(Comment, CommentZt)){
1063                Modified = TRUE;
1064                CommentSize = strlen(Comment);
1065            }
1066            if (EditComment)
1067  #else
1068            memcpy(Comment, (char *)CommentSec->Data+2, CommentSize);
1069  #endif
1070            {
1071                char EditFileName[PATH_MAX+5];
1072                strcpy(EditFileName, FileName);
1073                strcat(EditFileName, ".txt");
1074
1075                CommentSize = FileEditComment(EditFileName, Comment, CommentSize);
1076            }
1077        }
1078
1079        if (strcmp(Comment, (char *)CommentSec->Data+2)){
1080            // Discard old comment section and put a new one in.
1081            int size;
1082            size = CommentSize+2;
1083            free(CommentSec->Data);
1084            CommentSec->Size = size;
1085            CommentSec->Data = malloc(size);
1086            CommentSec->Data[0] = (uchar)(size >> 8);
1087            CommentSec->Data[1] = (uchar)(size);
1088            memcpy((CommentSec->Data)+2, Comment, size-2);
1089            Modified = TRUE;
1090        }
1091        if (!Modified){
1092            printf("Comment not modified\n");
1093        }
1094    }
1095
1096
1097    if (CommentSavefileName){
1098        Section_t * CommentSec;
1099        CommentSec = FindSection(M_COM);
1100
1101        if (CommentSec != NULL){
1102            char OutFileName[PATH_MAX+1];
1103            FILE * CommentFile;
1104
1105            // Make a relative name.
1106            RelativeName(OutFileName, CommentSavefileName, FileName);
1107
```

```
1108            CommentFile = fopen(OutFileName,"w");
1109            if (CommentFile){
1110                fwrite((char *)CommentSec->Data+2 ,CommentSec->Size-2, 1, CommentFile);
1111                fclose(CommentFile);
1112            }else{
1113                ErrFatal("Could not write comment file");
1114            }
1115        }else{
1116            printf("File '%s' contains no comment section\n",FileName);
1117        }
1118    }
1119
1120    if (ExifTimeAdjust || ExifTimeSet || DateSetChars || FileTimeToExif){
1121       if (ImageInfo.numDateTimeTags){
1122            struct tm tm;
1123            time_t UnixTime;
1124            char TempBuf[50];
1125            int a;
1126            Section_t * ExifSection;
1127            if (ExifTimeSet){
1128                // A time to set was specified.
1129                UnixTime = ExifTimeSet;
1130            }else{
1131                if (FileTimeToExif){
1132                    FileTimeAsString(ImageInfo.DateTime);
1133                }
1134                if (DateSetChars){
1135                    memcpy(ImageInfo.DateTime, DateSet, DateSetChars);
1136                    a = 1970;
1137                    sscanf(DateSet, "%d", &a);
1138                    if (a < 1970){
1139                        strcpy(TempBuf, ImageInfo.DateTime);
1140                        goto skip_unixtime;
1141                    }
1142                }
1143                // A time offset to adjust by was specified.
1144                if (!Exif2tm(&tm, ImageInfo.DateTime)) goto badtime;
1145
1146                // Convert to unix 32 bit time value, add offset, and convert back.
1147                UnixTime = mktime(&tm);
1148                if ((int)UnixTime == -1) goto badtime;
1149                UnixTime += ExifTimeAdjust;
1150            }
1151            tm = *localtime(&UnixTime);
1152
1153            // Print to temp buffer first to avoid putting null termination in destination.
1154            // snprintf() would do the trick, but not available everywhere (like FreeBSD 4.4)
1155            sprintf(TempBuf, "%04d:%02d:%02d %02d:%02d:%02d",
1156                tm.tm_year+1900, tm.tm_mon+1, tm.tm_mday,
```

```
1157                tm.tm_hour, tm.tm_min, tm.tm_sec);
1158
1159 skip_unixtime:
1160             ExifSection = FindSection(M_EXIF);
1161
1162             for (a = 0; a < ImageInfo.numDateTimeTags; a++) {
1163                 uchar * Pointer;
1164                 Pointer = ExifSection->Data+ImageInfo.DateTimeOffsets[a]+8;
1165                 memcpy(Pointer, TempBuf, 19);
1166             }
1167             memcpy(ImageInfo.DateTime, TempBuf, 19);
1168
1169             Modified = TRUE;
1170         }else{
1171             printf("File '%s' contains no Exif timestamp to change\n", FileName);
1172         }
1173     }
1174
1175     if (DeleteComments){
1176         if (RemoveSectionType(M_COM)) Modified = TRUE;
1177     }
1178     if (DeleteExif){
1179         if (RemoveSectionType(M_EXIF)) Modified = TRUE;
1180     }
1181     if (DeleteIptc){
1182         if (RemoveSectionType(M_IPTC)) Modified = TRUE;
1183     }
1184     if (DeleteXmp){
1185         if (RemoveSectionType(M_XMP)) Modified = TRUE;
1186     }
1187     if (DeleteUnknown){
1188         if (RemoveUnknownSections()) Modified = TRUE;
1189     }
1190
1191
1192     if (Modified){
1193         char BackupName[PATH_MAX+5];
1194         struct stat buf;
1195
1196         if (!Quiet) printf("Modified: %s\n",FileName);
1197
1198         strcpy(BackupName, FileName);
1199         strcat(BackupName, ".t");
1200
1201         // Remove any .old file name that may pre-exist
1202         unlink(BackupName);
1203
1204         // Rename the old file.
1205         rename(FileName, BackupName);
```

```
1206
1207            // Write the new file.
1208            WriteJpegFile(FileName);
1209
1210            // Copy the access rights from original file
1211            if (stat(BackupName, &buf) == 0){
1212                // set Unix access rights and time to new file
1213                struct utimbuf mtime;
1214                chmod(FileName, buf.st_mode);
1215
1216                mtime.actime = buf.st_mtime;
1217                mtime.modtime = buf.st_mtime;
1218
1219                utime(FileName, &mtime);
1220            }
1221
1222            // Now that we are done, remove original file.
1223            unlink(BackupName);
1224        }
1225
1226
1227        if (Exif2FileTime){
1228            // Set the file date to the date from the exif header.
1229            if (ImageInfo.numDateTimeTags){
1230                // Convert the file date to Unix time.
1231                struct tm tm;
1232                time_t UnixTime;
1233                struct utimbuf mtime;
1234              if (!Exif2tm(&tm, ImageInfo.DateTime)) goto badtime;
1235                UnixTime = mktime(&tm);
1236                if ((int)UnixTime == -1){
1237                    goto badtime;
1238                }
1239                mtime.actime = UnixTime;
1240                mtime.modtime = UnixTime;
1241
1242                if (utime(FileName, &mtime) != 0){
1243                    printf("Error: Could not change time of file '%s'\n",FileName);
1244                }else{
1245                    if (!Quiet) printf("%s\n",FileName);
1246                }
1247            }else{
1248                printf("File '%s' contains no Exif timestamp\n", FileName);
1249            }
1250        }
1251
1252        // Feature to rename image according to date and time from camera.
1253        // I use this feature to put images from multiple digicams in sequence.
1254
```

```
    if (RenameToDate){
        DoFileRenaming(FileName);
    }
    DiscardData();
    return;
badtime:
    printf("Error: Time '%s': cannot convert to Unix time\n",ImageInfo.DateTime);
    DiscardData();
}

//--------------------------------------------------------------------------
// complain about bad state of the command line.
//--------------------------------------------------------------------------
static void Usage (void)
{
    printf("Jhead is a program for manipulating settings and thumbnails in Exif jpeg headers\n"
           "used by most Digital Cameras.  v"JHEAD_VERSION" Matthias Wandel, Oct 5 2020.\n"
           "http://www.sentex.net/~mwandel/jhead\n"
           "\n");

    printf("Usage: %s [options] files\n", progname);
    printf("Where:\n"
           " files       path/filenames with or without wildcards\n"

           "[options] are:\n"
           "\nGENERAL METADATA:\n"
           "  -te <name> Transfer exif header from another image file <name>\n"
           "             Uses same name mangling as '-st' option\n"
           "  -dc        Delete comment field (as left by progs like Photoshop & Compupic)\n"
           "  -de        Strip Exif section (smaller JPEG file, but lose digicam info)\n"
           "  -di        Delete IPTC section (from Photoshop, or Picasa)\n"
           "  -dx        Delete XMP section\n"
           "  -du        Delete non image sections except for Exif and comment sections\n"
           "  -purejpg   Strip all unnecessary data from jpeg (combines -dc -de and -du)\n"
           "  -mkexif    Create new minimal exif section (overwrites pre-existing exif)\n"
           "  -ce        Edit comment field.  Uses environment variable 'editor' to\n"
           "             determine which editor to use.  If editor not set, uses VI\n"
           "             under Unix and notepad with windows\n"
           "  -cs <name> Save comment section to a file\n"
           "  -ci <name> Insert comment section from a file.  -cs and -ci use same naming\n"
           "             scheme as used by the -st option\n"
           "  -cl string Insert literal comment string\n"
           "  -zt        Trim exif header trailing zeroes (Nikon 1 wastes 30k that way)\n"

           "\nDATE / TIME MANIPULATION:\n"
           "  -ft        Set file modification time to Exif time\n"
           "  -dsft      Set Exif time to file modification time\n"
           "  -n[format-string]\n"
           "             Rename files according to date.  Uses exif date if present, file\n"
```

```
1304            "             date otherwise.  If the optional format-string is not supplied,\n"
1305            "             the format is mmdd-hhmmss.  If a format-string is given, it is\n"
1306            "             is passed to the 'strftime' function for formatting\n"
1307            "             %%d Day of month    %%H Hour (24-hour)\n"
1308            "             %%m Month number    %%M Minute    %%S Second\n"
1309            "             %%y Year (2 digit 00 - 99)        %%Y Year (4 digit 1980-2036)\n"
1310            "             For more arguments, look up the 'strftime' function.\n"
1311            "             In addition to strftime format codes:\n"
1312            "             '%%f' as part of the string will include the original file name\n"
1313            "             '%%i' will include a sequence number, starting from 1. You can\n"
1314            "             You can specify '%%03i' for example to get leading zeros.\n"
1315            "             This feature is useful for ordering files from multiple digicams to\n"
1316            "             sequence of taking.\n"
1317            "             The '.jpg' is automatically added to the end of the name.  If the\n"
1318            "             destination name already exists, a letter or digit is added to \n"
1319            "             the end of the name to make it unique.\n"
1320            "             The new name may include a path as part of the name.  If this path\n"
1321            "             does not exist, it will be created\n"
1322            "  -a         (Windows only) Rename files with same name but different extension\n"
1323            "             Use together with -n to rename .AVI files from exif in .THM files\n"
1324            "             for example\n"
1325            "  -ta<+|->h[:mm[:ss]]\n"
1326            "             Adjust time by h:mm forwards or backwards.  Useful when having\n"
1327            "             taken pictures with the wrong time set on the camera, such as when\n"
1328            "             traveling across time zones or DST changes. Dates can be adjusted\n"
1329            "             by offsetting by 24 hours or more.  For large date adjustments,\n"
1330            "             use the -da option\n"
1331            "  -da<date>-<date>\n"
1332            "             Adjust date by large amounts.  This is used to fix photos from\n"
1333            "             cameras where the date got set back to the default camera date\n"
1334            "             by accident or battery removal.\n"
1335            "             To deal with different months and years having different numbers of\n"
1336            "             days, a simple date-month-year offset would result in unexpected\n"
1337            "             results.  Instead, the difference is specified as desired date\n"
1338            "             minus original date.  Date is specified as yyyy:mm:dd or as date\n"
1339            "             and time in the format yyyy:mm:dd/hh:mm:ss\n"
1340            "  -ts<time>  Set the Exif internal time to <time>.  <time> is in the format\n"
1341            "             yyyy:mm:dd-hh:mm:ss\n"
1342            "  -tf file   Set the exif time to the modification time from another file\n"
1343            "  -ds<date>  Set the Exif internal date.  <date> is in the format YYYY:MM:DD\n"
1344            "             or YYYY:MM or YYYY\n"
1345
1346            "\nTHUMBNAIL MANIPULATION:\n"
1347            "  -dt        Remove exif integral thumbnails.  Typically trims 10k\n"
1348            "  -st <name> Save Exif thumbnail, if there is one, in file <name>\n"
1349            "             If output file name contains the substring \"&i\" then the\n"
1350            "             image file name is substitute for the &i.  Note that quotes around\n"
1351            "             the argument are required for the '&' to be passed to the program.\n"
1352  #ifndef _WIN32
```

```
1353            "             An output name of '-' causes thumbnail to be written to stdout\n"
1354    #endif
1355            "  -rt <name> Replace Exif thumbnail.  Can only be done with headers that\n"
1356            "             already contain a thumbnail.\n"
1357            "  -rgt[size] Regenerate exif thumbnail.  Only works if image already\n"
1358            "             contains a thumbnail.  size specifies maximum height or width of\n"
1359            "             thumbnail.  Relies on 'mogrify' programs to be on path\n"
1360
1361            "\nROTATION TAG MANIPULATION:\n"
1362            "  -autorot   Invoke jpegtran to rotate images according to Exif orientation tag\n"
1363            "             and clear Exif orientation tag\n"
1364            "             Note: Windows users must get jpegtran for this to work\n"
1365            "  -norot     Zero out the rotation tag.  This to avoid some browsers from\n"
1366            "             rotating the image again after you rotated it but neglected to\n"
1367            "             clear the rotation tag\n"
1368
1369            "\nOUTPUT VERBOSITY CONTROL:\n"
1370            "  -h         help (this text)\n"
1371            "  -v         even more verbose output\n"
1372            "  -q         Quiet (no messages on success, like Unix)\n"
1373            "  -V         Show jhead version\n"
1374            "  -exifmap   Dump header bytes, annotate.  Pipe thru sort for better viewing\n"
1375            "  -se        Suppress error messages relating to corrupt exif header structure\n"
1376            "  -c         concise output\n"
1377            "  -nofinfo   Don't show file info (name/size/date)\n"
1378
1379            "\nFILE MATCHING AND SELECTION:\n"
1380            "  -model model\n"
1381            "             Only process files from digicam containing model substring in\n"
1382            "             camera model description\n"
1383            "  -exonly    Skip all files that don't have an exif header (skip all jpegs that\n"
1384            "             were not created by digicam)\n"
1385            "  -quality x Only work on images with JPEG quality factor x or higher\n"
1386            "  -cmd command\n"
1387            "             Apply 'command' to every file, then re-insert exif and command\n"
1388            "             sections into the image. &i will be substituted for the input file\n"
1389            "             name, and &o (if &o is used). Use quotes around the command string\n"
1390            "             This is most useful in conjunction with the free ImageMagick tool. \n"
1391            "             For example, with my Canon S100, which suboptimally compresses\n"
1392            "             jpegs I can specify\n"
1393            "                jhead -cmd \"mogrify -quality 80 &i\" *.jpg\n"
1394            "             to re-compress a lot of images using ImageMagick to half the size,\n"
1395            "             and no visible loss of quality while keeping the exif header\n"
1396            "             Another invocation I like to use is jpegtran (hard to find for\n"
1397            "             windows).  I type:\n"
1398            "                jhead -cmd \"jpegtran -progressive &i &o\" *.jpg\n"
1399            "             to convert jpegs to progressive jpegs (Unix jpegtran syntax\n"
1400            "             differs slightly)\n"
1401            "  -orp       Only operate on 'portrait' aspect ratio images\n"
```

```
1402            "  -orl       Only operate on 'landscape' aspect ratio images\n"
1403    #ifdef _WIN32
1404            "  -r         No longer supported.  Use the ** wildcard to recurse directories\n"
1405            "             with instead.\n"
1406            "             examples:\n"
1407            "                 jhead **/*.jpg\n"
1408            "                 jhead \"c:\\my photos\\**\\*.jpg\"\n"
1409    #endif
1410
1411
1412    #ifdef MATTHIAS
1413            "\n"
1414            "  -cr        Remove comment tag (my way)\n"
1415            "  -ca        Add comment tag (my way)\n"
1416            "  -ar        Auto resize to fit in 1024x1024, but never less than half\n"
1417    #endif //MATTHIAS
1418
1419
1420            );
1421
1422        exit(EXIT_FAILURE);
1423    }
1424
1425
1426    //--------------------------------------------------------------------------
1427    // Parse specified date or date+time from command line.
1428    //--------------------------------------------------------------------------
1429    static time_t ParseCmdDate(char * DateSpecified)
1430    {
1431        int a;
1432        struct tm tm;
1433        time_t UnixTime;
1434
1435        tm.tm_wday = -1;
1436        tm.tm_hour = tm.tm_min = tm.tm_sec = 0;
1437
1438        a = sscanf(DateSpecified, "%d:%d:%d/%d:%d:%d",
1439                &tm.tm_year, &tm.tm_mon, &tm.tm_mday,
1440                &tm.tm_hour, &tm.tm_min, &tm.tm_sec);
1441
1442        if (a != 3 && a < 5){
1443            // Date must be YYYY:MM:DD, YYYY:MM:DD+HH:MM
1444            // or YYYY:MM:DD+HH:MM:SS
1445            ErrFatal("Could not parse specified date");
1446        }
1447        tm.tm_isdst = -1;
1448        tm.tm_mon -= 1;      // Adjust for unix zero-based months
1449        tm.tm_year -= 1900;  // Adjust for year starting at 1900
1450
```

```
1451        UnixTime = mktime(&tm);
1452        if (UnixTime == -1){
1453            ErrFatal("Specified time is invalid or out of range");
1454        }
1455
1456        return UnixTime;
1457    }
1458
1459    //--------------------------------------------------------------------------
1460    // The main program.
1461    //--------------------------------------------------------------------------
1462    int main (int argc, char **argv)
1463    {
1464        int argn;
1465        char * arg;
1466        progname = argv[0];
1467
1468        for (argn=1;argn<argc;argn++){
1469            arg = argv[argn];
1470            if (arg[0] != '-') break; // Filenames from here on.
1471
1472        // General metadata options:
1473            if (!strcmp(arg,"-te")){
1474                ExifXferScrFile = argv[++argn];
1475                DoModify |= MODIFY_JPEG;
1476            }else if (!strcmp(arg,"-dc")){
1477                DeleteComments = TRUE;
1478                DoModify |= MODIFY_JPEG;
1479            }else if (!strcmp(arg,"-de")){
1480                DeleteExif = TRUE;
1481                DoModify |= MODIFY_JPEG;
1482            }else if (!strcmp(arg,"-di")){
1483                DeleteIptc = TRUE;
1484                DoModify |= MODIFY_JPEG;
1485            }else if (!strcmp(arg,"-dx")){
1486                DeleteXmp = TRUE;
1487                DoModify |= MODIFY_JPEG;
1488            }else if (!strcmp(arg, "-du")){
1489                DeleteUnknown = TRUE;
1490                DoModify |= MODIFY_JPEG;
1491            }else if (!strcmp(arg, "-purejpg")){
1492                DeleteExif = TRUE;
1493                DeleteComments = TRUE;
1494                DeleteIptc = TRUE;
1495                DeleteUnknown = TRUE;
1496                DeleteXmp = TRUE;
1497                DoModify |= MODIFY_JPEG;
1498            }else if (!strcmp(arg,"-ce")){
1499                EditComment = TRUE;
```

```
1500            DoModify |= MODIFY_JPEG;
1501        }else if (!strcmp(arg,"-cs")){
1502            CommentSavefileName = argv[++argn];
1503        }else if (!strcmp(arg,"-ci")){
1504            CommentInsertfileName = argv[++argn];
1505            DoModify |= MODIFY_JPEG;
1506        }else if (!strcmp(arg,"-cl")){
1507            CommentInsertLiteral = argv[++argn];
1508            DoModify |= MODIFY_JPEG;
1509        }else if (!strcmp(arg,"-zt")){
1510            TrimExifTrailingZeroes = TRUE;
1511            DoModify |= MODIFY_JPEG;
1512        }else if (!strcmp(arg,"-mkexif")){
1513            CreateExifSection = TRUE;
1514            DoModify |= MODIFY_JPEG;
1515    // Output verbosity control
1516        }else if (!strcmp(arg,"-h") || !strcmp(arg,"--help")){
1517            Usage();
1518        }else if (!strcmp(arg,"-v")){
1519            ShowTags = TRUE;
1520        }else if (!strcmp(arg,"-q")){
1521            Quiet = TRUE;
1522        }else if (!strcmp(arg,"-V")){
1523            printf("Jhead version: "JHEAD_VERSION"\n");
1524            exit(0);
1525        }else if (!strcmp(arg,"-exifmap")){
1526            DumpExifMap = TRUE;
1527        }else if (!strcmp(arg,"-se")){
1528            SuppressNonFatalErrors = TRUE;
1529        }else if (!strcmp(arg,"-c")){
1530            ShowConcise = TRUE;
1531        }else if (!strcmp(arg,"-nofinfo")){
1532            ShowFileInfo = 0;
1533
1534    // Thumbnail manipulation options
1535        }else if (!strcmp(arg,"-dt")){
1536            TrimExif = TRUE;
1537            DoModify |= MODIFY_JPEG;
1538        }else if (!strcmp(arg,"-st")){
1539            ThumbSaveName = argv[++argn];
1540            DoModify |= READ_JPEG;
1541        }else if (!strcmp(arg,"-rt")){
1542            ThumbInsertName = argv[++argn];
1543            DoModify |= MODIFY_JPEG;
1544        }else if (!memcmp(arg,"-rgt", 4)){
1545            RegenThumbnail = 160;
1546            sscanf(arg+4, "%d", &RegenThumbnail);
1547            if (RegenThumbnail > 320){
1548                ErrFatal("Specified thumbnail geometry too big!");
```

```
1549            }
1550            DoModify |= MODIFY_JPEG;
1551
1552    // Rotation tag manipulation
1553        }else if (!strcmp(arg,"-autorot")){
1554            AutoRotate = 1;
1555            DoModify |= MODIFY_JPEG;
1556        }else if (!strcmp(arg,"-norot")){
1557            AutoRotate = 1;
1558            ZeroRotateTagOnly = 1;
1559            DoModify |= MODIFY_JPEG;
1560
1561    // Date/Time manipulation options
1562        }else if (!memcmp(arg,"-n",2)){
1563            RenameToDate = 1;
1564            DoModify |= READ_JPEG; // Rename doesn't modify file, so count as read action.
1565            arg+=2;
1566            if (*arg == 'f'){
1567                // Accept -nf, but -n does the same thing now.
1568                arg++;
1569            }
1570            if (*arg){
1571                // A strftime format string is supplied.
1572                strftime_args = arg;
1573                #ifdef _WIN32
1574                    SlashToNative(strftime_args);
1575                #endif
1576                //printf("strftime_args = %s\n",arg);
1577            }
1578        }else if (!strcmp(arg,"-a")){
1579            #ifndef _WIN32
1580                ErrFatal("Error: -a only supported in Windows version");
1581            #else
1582                RenameAssociatedFiles = TRUE;
1583            #endif
1584        }else if (!strcmp(arg,"-ft")){
1585            Exif2FileTime = TRUE;
1586            DoModify |= MODIFY_ANY;
1587        }else if (!memcmp(arg,"-ta",3)){
1588            // Time adjust feature.
1589            int hours, minutes, seconds, n;
1590            minutes = seconds = 0;
1591            if (arg[3] != '-' && arg[3] != '+'){
1592                ErrFatal("Error: -ta must be followed by +/- and a time");
1593            }
1594            n = sscanf(arg+4, "%d:%d:%d", &hours, &minutes, &seconds);
1595
1596            if (n < 1){
1597                ErrFatal("Error: -ta must be immediately followed by time");
```

```
1598            }
1599            if (ExifTimeAdjust) ErrFatal("Can only use one of -da or -ta options at once");
1600            ExifTimeAdjust = hours*3600 + minutes*60 + seconds;
1601            if (arg[3] == '-') ExifTimeAdjust = -ExifTimeAdjust;
1602            DoModify |= MODIFY_JPEG;
1603        }else if (!memcmp(arg,"-da",3)){
1604            // Date adjust feature (large time adjustments)
1605            time_t NewDate, OldDate = 0;
1606            char * pOldDate;
1607            NewDate = ParseCmdDate(arg+3);
1608            pOldDate = strstr(arg+1, "-");
1609            if (pOldDate){
1610                OldDate = ParseCmdDate(pOldDate+1);
1611            }else{
1612                ErrFatal("Must specify second date for -da option");
1613            }
1614            if (ExifTimeAdjust) ErrFatal("Can only use one of -da or -ta options at once");
1615            ExifTimeAdjust = NewDate-OldDate;
1616            DoModify |= MODIFY_JPEG;
1617        }else if (!memcmp(arg,"-dsft",5)){
1618            // Set exif time to the timestamp of the file.
1619            FileTimeToExif = TRUE;
1620            DoModify |= MODIFY_JPEG;
1621        }else if (!memcmp(arg,"-ds",3)){
1622            // Set date feature
1623            int a;
1624            // Check date validity and copy it.  Could be incompletely specified.
1625            strcpy(DateSet, "0000:01:01");
1626            for (a=0;arg[a+3];a++){
1627                if (isdigit(DateSet[a])){
1628                    if (!isdigit(arg[a+3])){
1629                        a = 0;
1630                        break;
1631                    }
1632                }else{
1633                    if (arg[a+3] != ':'){
1634                        a=0;
1635                        break;
1636                    }
1637                }
1638                DateSet[a] = arg[a+3];
1639            }
1640            if (a < 4 || a > 10){
1641                ErrFatal("Date must be in format YYYY, YYYY:MM, or YYYY:MM:DD");
1642            }
1643            DateSetChars = a;
1644            DoModify |= MODIFY_JPEG;
1645        }else if (!memcmp(arg,"-ts",3)){
1646            // Set the exif time.
```

```
1647            // Time must be specified as "yyyy:mm:dd-hh:mm:ss"
1648            char * c;
1649            struct tm tm;
1650
1651            c = strstr(arg+1, "-");
1652            if (c) *c = ' '; // Replace '-' with a space.
1653
1654            if (!Exif2tm(&tm, arg+3)){
1655                ErrFatal("-ts option must be followed by time in format yyyy:mm:dd-hh:mm:ss\n"
1656                        "Example: jhead -ts2001:01:01-12:00:00 foo.jpg");
1657            }
1658
1659            ExifTimeSet  = mktime(&tm);
1660
1661            if ((int)ExifTimeSet == -1) ErrFatal("Time specified is out of range");
1662            DoModify |= MODIFY_JPEG;
1663
1664        }else if (!memcmp(arg,"-tf",3)){
1665            // Set the exif time to the modification time from another file.
1666            struct stat stat_buf;
1667            if (stat(argv[++argn], &stat_buf) == 0){
1668                ExifTimeSet = stat_buf.st_mtime;
1669            }else{
1670                ErrFatal("Could not read file");
1671            }
1672            DoModify |= MODIFY_JPEG;
1673
1674    // File matching and selection
1675        }else if (!strcmp(arg,"-model")){
1676            if (argn+1 >= argc) Usage(); // No extra argument.
1677            FilterModel = argv[++argn];
1678        }else if (!strcmp(arg,"-quality")){
1679            if (argn+1 >= argc) Usage(); // No extra argument.
1680            if (sscanf(argv[++argn], "%d", &FilterQuality) != 1){
1681                Usage();
1682            }
1683        }else if (!strcmp(arg,"-proc")){
1684            sscanf(argv[++argn], "%d", &ProcessOnly);
1685            if (ProcessOnly < 0 || ProcessOnly > 2){
1686                ErrFatal("-proc must be followed by a number 0-2");
1687            }
1688        }else if (!strcmp(arg,"-exonly")){
1689            ExifOnly = 1;
1690        }else if (!strcmp(arg,"-orp")){
1691            PortraitOnly = 1;
1692        }else if (!strcmp(arg,"-orl")){
1693            PortraitOnly = -1;
1694        }else if (!strcmp(arg,"-cmd")){
1695            if (argn+1 >= argc) Usage(); // No extra argument.
```

```
1696            ApplyCommand = argv[++argn];
1697            DoModify |= MODIFY_ANY;
1698
1699 #ifdef MATTHIAS
1700        }else if (!strcmp(arg,"-ca")){
1701            // Its a literal comment.  Add.
1702            AddComment = argv[++argn];
1703            DoModify |= MODIFY_JPEG;
1704        }else if (!strcmp(arg,"-cr")){
1705            // Its a literal comment.  Remove this keyword.
1706            RemComment = argv[++argn];
1707            DoModify |= MODIFY_JPEG;
1708        }else if (!strcmp(arg,"-ar")){
1709            AutoResize = TRUE;
1710            ShowConcise = TRUE;
1711            ApplyCommand = (char *)1; // Must be non null so it does commands.
1712            DoModify |= MODIFY_JPEG;
1713 #endif // MATTHIAS
1714        }else{
1715            printf("Argument '%s' not understood\n",arg);
1716            printf("Use jhead -h for list of arguments\n");
1717            exit(-1);
1718        }
1719        if (argn >= argc){
1720            // Used an extra argument - because the last argument
1721            // used up an extr argument.
1722            ErrFatal("Extra argument required");
1723        }
1724    }
1725    if (argn == argc){
1726        ErrFatal("No files to process.  Use -h for help");
1727    }
1728
1729    if (ThumbSaveName != NULL && strcmp(ThumbSaveName, "&i") == 0){
1730        printf("Error: By specifying \"&i\" for the thumbail name, your original file\n"
1731               "       will be overwritten.  If this is what you really want,\n"
1732               "       specify -st \"./&i\"  to override this check\n");
1733        exit(0);
1734    }
1735
1736    if (RegenThumbnail){
1737        if (ThumbSaveName || ThumbInsertName){
1738            printf("Error: Cannot regen and save or insert thumbnail in same run\n");
1739            exit(0);
1740        }
1741    }
1742
1743    if (EditComment){
1744        if (CommentSavefileName != NULL || CommentInsertfileName != NULL){
```

```
1745                printf("Error: Cannot use -ce option in combination with -cs or -ci\n");
1746                exit(0);
1747            }
1748        }
1749
1750
1751        if (ExifXferScrFile){
1752            if (FilterModel || ApplyCommand){
1753                ErrFatal("Error: Filter by model and/or applying command to files\n"
1754                "   invalid while transferring Exif headers");
1755            }
1756        }
1757
1758        FileSequence = 0;
1759        for (;argn<argc;argn++){
1760            FilesMatched = FALSE;
1761
1762            #ifdef _WIN32
1763                SlashToNative(argv[argn]);
1764                // Use my globbing module to do fancier wildcard expansion with recursive
1765                // subdirectories under Windows.
1766                MyGlob(argv[argn], ProcessFile);
1767            #else
1768                // Under linux, don't do any extra fancy globbing - shell globbing is
1769                // pretty fancy as it is - although not as good as myglob.c
1770                ProcessFile(argv[argn]);
1771            #endif
1772
1773            if (!FilesMatched){
1774                fprintf(stderr, "Error: No files matched '%s'\n",argv[argn]);
1775            }
1776        }
1777
1778        if (FileSequence == 0){
1779            return EXIT_FAILURE;
1780        }else{
1781            return EXIT_SUCCESS;
1782        }
1783    }
```