

Forward

The CiderTag utility is a Windows Command-line Utility for facilitating the porting of Apple II programs and files from Windows to the Apple II.

It does so by making a copy of the original file, converting what needs to be converted (at the user's discretion). The copy is automatically named using the same name as the original file, appended by Apple II file system information which follows a naming convention called CiderPress File Attribute Preservation Tags developed by Andy McFadden, the author of [CiderPress](http://ciderpress.sourceforge.net/)

For those unfamiliar with their use, Andy's tutorial can be found at the following link:

<http://ciderpress.sourceforge.net/tutorial/>

If you develop for the Apple II under Windows, or otherwise work with disk images to run in emulators like [AppleWin](http://applewin.sourceforge.net/) and/or work with the porting of Apple II files to CF cards and other media for use with present-day Apple II cards like Rich Dreher's [CFFA3000](http://cfra3000.com/) you are likely familiar with CiderPress.

And if so, the CiderTag utility may prove useful to you in "massaging" and pre-processing your Apple II files in ways that aren't necessarily straight-forward and otherwise difficult without a lot of "fiddling" and bother.

Why I Wrote CiderTag



I am "on-record" and "guilty as charged" in saying that File Attribute Preservation Tags are "messy" because they make a Windows directory look ugly and cryptic. So now let me go "on-record" as saying that Andy's tags are not only a brilliant idea and well-worthy of the standard they have become, but also dead-dumb simple to program and to understand and to use.

Many years ago when the Apple II was still in wide-use in the late '80's and into the '90's, I used a serial cable to port my Apple II files between my IBM-PC and my Apple II. I was programming in Aztec C65 then (as I do now) but back then it was not exactly a hobby.

The Aztec C65 linker appends a binary header to every program that it creates, even to ProDOS SYS programs, so to facilitate my modem transfers of my programming I wrote a utility which just lobbed-off these seemingly useless headers. Fast-forward to the other

day... I've been using CiderPress for quite some years to get my latest programs to the Apple II and I finally asked myself why I was still using the old utility that I wrote so long ago to behead my files and then manually setting their attributes in CiderPress.

So I pulled-out my Windows compiler and very shortly thereafter CiderTag was "born". Like most programmers I began to tweak my new creation and to consider how other folks might want to use it. Despite my long-time resistance to "feature-creep", the potential benefits of my tweaks took over. A few hours later I thought I was done. But further testing revealed some additional tweaks that would be desirable like preserving file times of input files and preserving of "Camel Cased" filenames.

But with the writing of this document, the "tweaks" have stopped and CiderTag is complete and ready to be unleashed on an unsuspecting Planet filled with potential Apple II (and CiderPress) users.

In other words, CiderTag was written just for the "funnery" of it. Despite that, I hope it will be of some use to anyone who may want to use such a thing.

How to get CiderTag

This download is intended to be overlaid over the current AppleX distribution for anyone who wants to build the programs, but can also simply be downloaded, and used as-is. Executables and Disk Images are included. The main topic is "retro" Serial Communications in Aztec C65, but included are many others. This download also includes CiderTag.

User Documentation:

http://www.aztecmuseum.ca/extras/AppleX_UTILITY.pdf

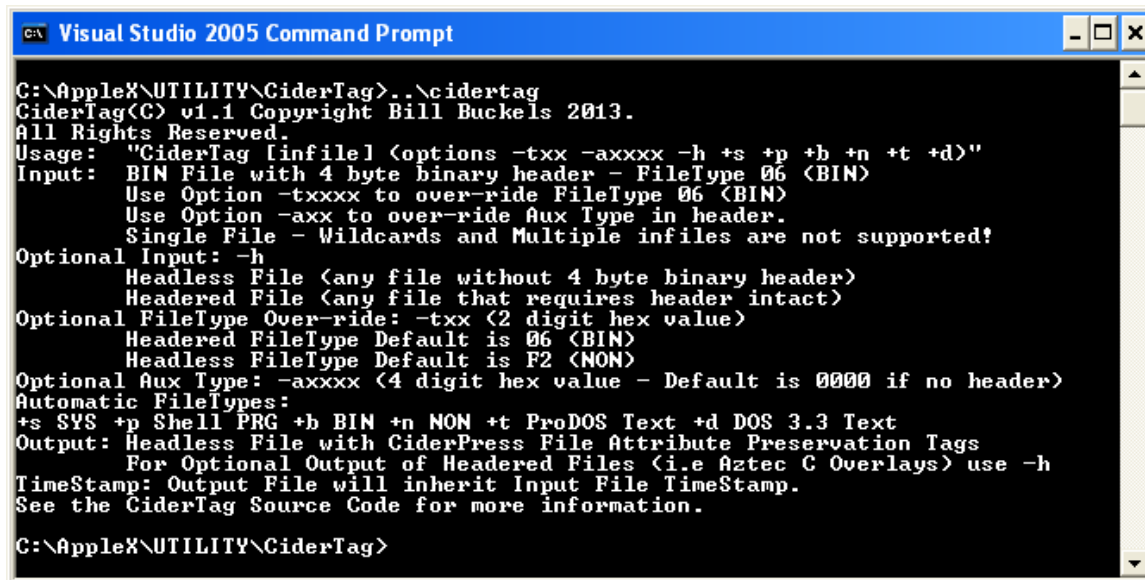
Programmer Notes:

http://www.aztecmuseum.ca/extras/AppleX_UTILITY_ReadMe.txt

Download it here:

http://www.aztecmuseum.ca/extras/AppleX_UTILITY.zip





```
C:\AppleX\UTILITY\CiderTag>.. \cidertag
CiderTag(C) v1.1 Copyright Bill Buckels 2013.
All Rights Reserved.
Usage: "CiderTag [infile] <options -txx -axxxx -h +s +p +b +n +t +d>"
Input:  BIN File with 4 byte binary header - FileType 06 <BIN>
        Use Option -txxxx to over-ride FileType 06 <BIN>
        Use Option -axx to over-ride Aux Type in header.
        Single File - Wildcards and Multiple infiles are not supported!
Optional Input: -h
        Headless File <any file without 4 byte binary header>
        Headered File <any file that requires header intact>
Optional FileType Over-ride: -txx <2 digit hex value>
        Headered FileType Default is 06 <BIN>
        Headless FileType Default is F2 <NON>
Optional Aux Type: -axxxx <4 digit hex value - Default is 0000 if no header>
Automatic FileTypes:
+s SYS +p Shell PRG +b BIN +n NON +t ProDOS Text +d DOS 3.3 Text
Output: Headless File with CiderPress File Attribute Preservation Tags
        For Optional Output of Headered Files (i.e Aztec C Overlays) use -h
TimeStamp: Output File will inherit Input File TimeStamp.
See the CiderTag Source Code for more information.

C:\AppleX\UTILITY\CiderTag>
```

(note: if you are having trouble reading the above, zoom to 200% or so)

Usage Overview

It goes without saying that knowledge of the Apple II File System is needed to use a sophisticated tool like CiderPress. So when it comes to using the little CiderTag utility as a helper tool, you will hopefully already have been seasoned and educated by “Professor Andy” and indoctrinated into the mysteries of all that sort of CiderPress good-stuff.

So we’ll start with a quick over-view of CiderTag’s functionality.

CiderTag runs in a Console Window under Windows. It is not an MS-DOS Utility, so it can’t be run in an MS-DOS emulator like [DOSBox](#).

When you start CiderTag without arguments (type “CiderTag” and press “Enter” with CiderTag “on-path”) the usage screen shown above will be presented.

Single File

As shown above, CiderTag operates on one file at a time. Since it is trivial to write a cmd script for Windows that processes a group of files by calling CiderTag repetitively, the omission of a WildCard option presents no great hardship. After all, folks who use command line tools write scripts to call them all the time:

@echo off

for %%f in (*.BIN) do call CiderTag %%f -t06 -a4000 -h

Setting of FileTypes and Auxiliary Types and Operating on Headless Files

As shown in the script above, the setting of a FileType, an Auxiliary Type, and operating on a file without a header are straight-forward options.

Files that Already Have Tags

CiderTag will skip converting a file if it has a tag already.

Mutually Exclusive and Duplicate Arguments

Obviously you can break almost anything. CiderTag is no exception. So let's say you keep repeating a bunch of different mutually exclusive options on the CiderTag command line; what will happen? Or let's say you type in multiple filenames?

To begin with, I let you do that, if that's what you want to do. You may use this program for whatever you wish as long as you agree that I have no warranty or liability obligations whatsoever from said use.

Secondly, CiderTag will overwrite files without prompting, so if you didn't do-it correctly the first time you can just do-it over. CiderTag has no "training-wheels".

But the real answer to the question is that the last FileType and Auxiliary Type entered will be used, or the last input filename entered will be used. However, since GIGO applies here, as with any programmer's or power-user's utility, the best advice I can give you is always proceed with caution. 'nuff said.

Date and Time Stamps

The Date and Time of the input file will be "stamped" on the output file. This is just the way I have programmed CiderTag, and this functionality is not optional.

Automatic FileTypes

These are "quick-switches" that I put into CiderTag to avoid much typing for myself. They are for the most common things that I do with an Aztec C65 program.

So now we move-on to the CiderTag Source Code.



CiderTag Source Code

CiderTag is written in the C Programming Language and not C++. It is a very short and trivial program, and is pretty-well commented. At the top of the program I placed the usual header info with short description:

```
// -----  
// System      : Win32  
// Program     : CiderTag.c  
// Description  : A C Programmers Tool for converting files for the Apple II  
//               Create Headless Files with File Attribute Preservation Tags  
//               from Binary (and other) Files  
//               Handles long filenames (of course).  
//               Not done to any great brilliance.  
//               No Wildcard Support.  
// Written by   : Bill Buckels  
// Date Written : November 10, 2013  
// Revision    : 1.0 Initial Release  
//             : 1.1 TimeStamping added and general polishing.  
//  
// Licence     : You may use this program for whatever you wish as long  
//               as you agree that Bill Buckels has no warranty or  
//               liability obligations whatsoever from said use.  
// -----
```

Below the header info I placed defines and includes. Since this is a console app, MicroSoft C requires me to explicitly abdicate them of responsibility of using what are supposedly unsafe and deprecated function calls by the following blessing:

```
#ifndef _CRT_SECURE_NO_DEPRECATED  
#define _CRT_SECURE_NO_DEPRECATED  
#endif  
  
#include <stdlib.h>  
#include <stdio.h>  
#include <string.h>  
#include <io.h>  
#include <time.h>  
#include <sys/utime.h>
```

Following my blessing and commissioning, I have placed some global buffers. What they are is pretty obvious:

```
char buf[_MAX_PATH],  
      ftype[30],  
      faddr[30],  
      infile[_MAX_PATH],  
      outfile[_MAX_PATH];
```

My function for printing the usage is pretty obvious too. It's just a series of put-strings:

```

void pusage()
{
puts("Usage:  \"CiderTag [infile] (options -txx -axxxx -h +s +p +b +n +t
+d)\");
puts("Input:  BIN File with 4 byte binary header - FileType 06 (BIN)");
puts("        Use Option -txxxx to over-ride FileType 06 (BIN)");
puts("        Use Option -axx to over-ride Aux Type in header.");
puts("        Single File - Wildcards and Multiple infiles are not supported!");
puts("Optional Input: -h");
puts("        Headless File (any file without 4 byte binary header)");
puts("        Headered File (any file that requires header intact)");
puts("Optional FileType Over-ride: -txx (2 digit hex value)");
puts("        Headered FileType Default is 06 (BIN)");
puts("        Headless FileType Default is F2 (NON)");
puts("Optional Aux Type: -axxxx (4 digit hex value - Default is 0000 if no
header)");
puts("Automatic FileTypes:");
puts("+s SYS +p Shell PRG +b BIN +n NON +t ProDOS Text +d DOS 3.3 Text");
puts("Output:  Headless File with CiderPress File Attribute Preservation Tags");
puts("        For Optional Output of Headered Files (i.e Aztec C Overlays) use
-h");
puts("TimeStamp: Output File will inherit Input File TimeStamp.");
puts("See the CiderTag Source Code for more information.");
}

```

I also made a little function that converts strings to lower-case and returns the string length while it is at it. I use it when I get the command-line args and parse the options and just want to keep this-out of the main code to make things more readable down there:

```

int lcase(char *ptr)
{
    int i, len;
    char c;

    len = strlen(ptr);
    for (i=0;i<len;i++) {
        c = ptr[i];
        ptr[i] = tolower(c);
    }
    return len;
}

```

The following function is used to date-stamp the output file. It's really only a wrapper for utime and is called from main as well (everything is):

```

/* use the _utime function to copy file times from the input file
to the output file */
int TimeStamp(char *fname, time_t actime, time_t modtime)
{
    /* structure for call to _utime */
    struct _utimbuf timbuf, *ptimbuf = NULL;

    timbuf.actime = actime;
    timbuf.modtime = modtime;

    ptimbuf = (struct _utimbuf *)&timbuf;
}

```

```

    /* set the times in the target file */
    if( _utime(fname, ptimbuf ) == -1 ) return -2;
    return 0;
}

```

This little function below checks input filenames to ensure that we skip files with tags. Also called from main:

```

int AlreadyDone(char *fname)
{
    char *ptr;
    int idx, jdx=999;

    for (idx = 0; fname[idx] != (char)0; idx++) {
        if (fname[idx] == '#') jdx = idx;
    }
    if (jdx == 999) return 0;
    ptr = (char *)&fname[jdx];
    idx = strlen(ptr);
    if (idx == 7) return -1;
    return 0;
}

```

Main begins with the declarations of the automatics (stack variables) that I use to process the input file. I set some defaults here as well. Note that I set my return status to an error (non-zero) until I am done. Note also that stripping of header info is the default unless otherwise disabled. The naming is probably descriptive enough for you to get the idea here:

```

int main(int argc, char **argv)
{
    FILE *fp, *fp2;

    long count = 0, target, findhandle;
    /* structure for file find to get time stamps and preserve
       CamelCasing (upper and lower case letters) from input file
       for output file */
    struct _finddata_t wild_card;

    int status=1, strip=1, len, addr=0, i;
    unsigned char c, msk = 0, txt = 0;

    /* timestamp the output file with the times from the input file */
    time_t actime;          /* access time */
    time_t modtime;         /* modification time */

```

Now we are good to go and if no args we just print the usage after we print the banner:

```

    /* no quiet mode for this utility... for diagnostic reasons.
       if they screw-up with their options then wonder why their files
       didn't convert properly this is the kindest thing to do... */

    puts("CiderTag(C) v1.1 Copyright Bill Buckels 2013.");
    puts("All Rights Reserved.");
    if (argc < 2) {

```

```

    pusage();
}

```

If we do have some command-line args we are open for business so let's see what our user has entered...

```

else {
    /* get input file and options */
    /* valid options are echoed to the screen,
       invalid options are ignored and defaults are used instead */
    faddr[0] = ftype[0] = infile[0] = (char)0;
    for (i=1;i<argc;i++) {
        strcpy(buf,argv[i]);
        c = buf[0];
        if (c == '-') {
            len = lcase((char *)&buf[0]);
            c = buf[1];
            switch(c) {
                case 'h': if (len != 2) break;
                           puts("-h Do Not Strip Header!");
                           strip = 0;
                           break;
                case 't': if (len != 4) break;
                           strcpy(ftype,(char *)&buf[2]);
                           printf("-t FileType %s\n",ftype);
                           break;
                case 'a': if (len != 6) break;
                           strcpy(faddr,(char *)&buf[2]);
                           printf("-a Aux Type %s\n",faddr);
                           break;
            }
            continue;
        }
        if (c == '+') {
            len = lcase((char *)&buf[0]);
            if (len > 2) continue;
            c = buf[1];
            switch(c) {
                case 's': strcpy(ftype,"ff");
                           printf("+s FileType %s\n",ftype);
                           break;
                case 'p': strcpy(ftype,"f8");
                           printf("+p FileType %s\n",ftype);
                           break;
                case 'b': strcpy(ftype,"06");
                           printf("+b FileType %s\n",ftype);
                           break;
                case 'n': strcpy(ftype,"f2");
                           printf("+n FileType %s (NON)\n",ftype);
                           puts("    Header if any remains intact!");
                           strip = 0;
                           break;
                case 't': strcpy(ftype,"04");
                           printf("+t FileType %s\n",ftype);
                           puts("    No Header. Convert to ProDOS Text!");
                           strip = 0;
                           txt = 1;
                           break;
                case 'd': strcpy(ftype,"04");
                           printf("+d FileType %s\n",ftype);
                           puts("    No Header. Convert to DOS 3.3 Text!");
                           strip = 0;
            }
        }
    }
}

```

```

        msk = 0x80;
        txt = 1;
        break;
    }
    continue;
}

/* the following is our first test to see if a valid file name
has been entered as an input file... */
if (AlreadyDone(buf)!=0) {
    printf("Skipping %s. Attribute Flags already exist!\n",buf);
    return status;
}

if((findhandle = _findfirst(buf, &wild_card)) < 11) continue;
/* get the times from the source file */
actime = wild_card.time_access;
modtime = wild_card.time_write;
/* Windows is not case-sensitive when it comes to opening
files, and since this is a commandline utility it is
unlikely the user typed the input filename using mixed
case.

But for Aesthetics, get the proper mixed case letters for
the output filename from the file system. */

strcpy(outfile,wild_card.name);

/* release the handle */
_findclose(findhandle);

/* echo the input file to the screen */
/* note that I have not used the filename returned from the
file system for the input file and instead I have stuck
with the filename that the user entered. From my view this
is a good thing. If the user entered a pathed name, this
utility will still likely work and the output file will be
dropped in the current directory. But since wildcards are
not supported, if the user has entered wildcards, this
utility will fail when we try to open the input file. This
is also a good thing from my view since supporting
wildcards is beyond the scope of CiderTag. */

strcpy(infile,buf);
printf("Infile   : %s\n",infile);

}

/* if the filetype for the output filename tag has not been set
by now, set-it to a bin file unless we are not stripping the
header, in which case set it to a NON type and let them
know... */
if (ftype[0] == (char)0) {
    if (strip == 1) {
        strcpy(ftype,"06");
        printf("+b FileType %s\n",ftype);
    }
    else {
        strcpy(ftype,"f2");
        printf("+n FileType %s (NON)\n",ftype);
    }
}
}

```

```

/* if we are stripping the header, let them know */
if (strip == 1) puts("+h Strip Header!");

```

We have finished getting our arguments and doing our initial set-up by the time we get to here. Now I use one of my favorite C constructs... “the forever loop” but I don’t really loop because I break unconditionally at the bottom. This construct is great for testing a set of conditions prior to a nested process loop... note the breaks every-time I find something I don’t like...

```

for(;;) {
    /* they may have screwed-up entering the input file name...
       or some other issue so check for common errors
       and let them know... */
    if (infile[0] == (char)0) {
        pusage();
        puts("No Input File!");
        break;
    }
    fp = fopen(infile,"rb");
    if (fp == NULL) {
        perror(infile);
        break;
    }
    target = _filelength(_fileno(fp));
    if (strip == 1) target -=4;
    if (target < 1L) {
        puts("Input File is too small!");
        fclose(fp);
        break;
    }

    if (strip == 1) {
        /* if we are stripping the header, use the first
           2 bytes to create the Aux Type (load address) */
        c=fgetc(fp);
        addr = (int)fgetc(fp);
        addr = (int)((addr << 8) | c);
        fgetc(fp);
        fgetc(fp);
    }

    /* if they did not previously over-ride the Aux Type
       on the command-line then use the value from the
       header or the initial value of 0 for the Aux Type
       and let them know what it is... */
    if (faddr[0] == (char)0) {
        sprintf(faddr,"%04x",addr);
        lcase(faddr);
        printf("+a Aux Type %s\n",faddr);
    }

    /* We already have the basename of the input file in the
       output filename buffer.
       Concatenate the file attribute preservation tags
       to the input file name after the pound-sign separator
       to create the output filename, try to open it,
       then let them know what the output file will be called. */

    strcat(outfile,"#");
    strcat(outfile,ftype);
}

```

```

strcat(outfile,faddr);

fp2= fopen(outfile,"wb");

if (fp2 == NULL) {
    fclose(fp);
    perror(outfile);
    break;
}

printf("Outfile : %s\n",outfile);

/* read the input file a byte at a time.

Text File Notes:

if we are explicitly outputting an ascii text file
strip the MS-DOS line feeds if any... and strip the
hi-bits.

I am not bothering to expand tabs for this version.

If we are explicitly outputting to a DOS 3.3 Text File,
set the hi-bits except for the NUL characters.

I am not bothering with Unix text at all. This is not a
Unix text conversion utility nor have I a desire to make
it one.

However, I am also not converting any text to Apple II
text unless explicitly asked to do so.

The Aztec C DOS 3.3 Shell does not accept DOS 3.3 text for
Shell Scripts is one reason I don't just blindly convert.
Some things are best left alone.

As far as RATF's go, for DOS 3.3 the -d switch should be
used, and no Aux Type specified, since it is meaningless.

For ProDOS RATF's the -t switch should never be used...
but the -h switch should be used and the -a Aux Type
should be used to specify the record length in hex.

Anyone working with RATF's in Windows then porting them
back to the Apple II had best know their stuff.

*/

```

We are still in “the forever loop” so everything must be ok so far. So now we go into the nested process loop. This is nothing more than a byte for byte binary copy of the file data except as noted below:

```

while(count<target)
{
    c=fgetc(fp);

    /* I separated the explicit text file conversion
       from the simple verbatim bytecopy routine */
    if (txt == 1) {
        count++;
        /* Apple II text files are 7 bit ascii so standard

```

```

        practice is to preserve file integrity by
        stripping the hi-bits on the way over to the
        Apple II... */
    c &= 0x7f;
    /* skip line-feeds */
    if (c == (unsigned char)10) continue;

    /* NUL characters are valid in Apple II
       text files */
    if (c == (char) 0) {
        fputc(c,fp2);
        continue;
    }
    /* if we are converting to ProDOS text, the msk
       will be 0, so no harm done. But if we are
       converting to DOS 3.3 text, the msk will be
       0x80 and the hi-bits will be set. */
    fputc((c | msk),fp2);
    continue;
}

    fputc(c,fp2);
    count++;
}

fclose(fp);
fclose(fp2);
puts("Done!");
/* return 0 if written successfully */
status = 0;

/* if we were able to write the output file, it is unlikely
   that we will fail to stamp the file times from the input
   file. but I have put an error check here anyway... */
if (TimeStamp(outfile,actime,modtime) != 0)
    printf("TimeStamping of %s failed!\n.",outfile);

break;
}
}

```

As you can see we are done. We broke unconditionally out of “the forever loop” when we reached the bottom and now we can return safe and sound to the Windows console to await further orders:

```

    return (status);
}

```

End of Listing



Closing Remarks

Utilities for the Windows Console like CiderTag are pretty easy to knock together in the C Programming Language. Admittedly, at 80K, CiderTag seems a bit large when compared to an Aztec C65 program for the Apple II, but compared to some of the “bloatware” that we see these days occupying entire DVD’s or even the size of this document, they are tiny for sure.

Apple II Links

Knowledge of the Apple II File System is essential for the use of this utility.



Here are some links that you may find useful:



DOS 3.3 disks can be read and edited at the track and sector level from either DOS 3.3 or ProDOS. Aztec C65 version 3.2b now supports this functionality. The following programs provide the basis to build Aztec C65 utilities that could be used to copy files from DOS 3.3 disks to ProDOS, or to create DOS 3.3 disk images (and other disk image types) on ProDOS media, and for the many other potential uses of the DOS 3.3 RWTS call and the ProDOS READBLOCK and WRITEBLOCK MLI calls.

Read about it here:

http://www.aztec-museum.ca/extras/chtype_DOS33.pdf

Download the working programs, disk images and source code here.

1. Changing DOS 3.3 FileTypes in an Aztec C65 Version 3.2b DOS 3.3 Program

DOS 3.3 RWTS example:

<http://www.aztec-museum.ca/extras/READBLK.zip>

If you have Apple33 installed, unzip into the AppleX\PROJECTS directory.

2. Changing DOS 3.3 FileTypes in an Aztec C65 Version 3.2b ProDOS 8 Program

ProDOS 8 READBLOCK equivalent example:

<http://www.aztecmuseum.ca/extras/CHTYPE.zip>

If you have AppleX installed, unzip into the AppleX\PROGRAMS directory.

The chtype and chtype33 example programs while relatively simple and straight-forward are more than simply demo programs. They actively change DOS 3.3 file types so use with caution.

Other related DOS 3.3 downloads:

The Aztec C65 DIR33 Project for Apple II DOS 3.3 - RWTS “revealed”:

<http://www.aztecmuseum.ca/extras/DIR33.zip>

<http://www.aztecmuseum.ca/extras/WorkingWithFiles.txt>

Other related ProDOS downloads:

<http://www.aztecmuseum.ca/UTL.zip>

CHANGE.DSK

CHMOD - change file permissions

CHTYPE - change file types

TOUCH - change file dates

LIST.DSK

FI - List ProDOS 8 file info (try it)

LM - List Multiple Files (try it)

All the Best,

Bill Buckels

bbuckels@mts.net

November 11, 2013