```
/*****************************************************************************
 *      G E N C C P M -  generates an operating system
 *
 *      Written by Bill Fitler Nov. 1982
 *
 * GENCCPM is designed to edit the system image for the operating system
 * and construct an initialized, loadable image for execution.
 * GENCCPM was originally designed for Concurrent CP/M-86 v2.0, and was
 * modelled after the old GENSYS.
 *
 * Revision history:
 *      23 Feb 83 whf: add patch module
 *      25 Jan 83 whf: combine all modules for easier maintenance
 *      20 Jan 83 whf: handles SYSDAT overflow
 *      19 Nov 82 whf: added BDOS buffering generation.
 *
 *****************************************************************************/


/*********************************/
/* for serialization purposes: */
char *copynote = "COPYRIGHT (C) 1983, DIGITAL RESEARCH 654321 $";
/*****************************************************/
/* up front for patching: Note the internal EOS (0) */
char *clearx = "\033H\033E\0\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n";
/*****************************************************/

/****** include files ********/
#define MAINMODULE
#define CCPM
#include <genccpm.h>
/*******************************/

#ifdef CCPM
BYTE *edsysver = "GENCCPM  v3.0  [3/7/83]\n";
#endif
#ifdef MPM
BYTE *edsysver = "GENMPM  vX.X  [??/??/??]\n";
#endif

BYTE mainmtitle[50] = "";               /* room for title of main menu */
MENU *mainmenu = NULLPTR;               /* pointers to main menu items    */
MENU *xiosmenu = NULLPTR;               /* pointers to XIOS menu items */
MENU *memmenu=NULLPTR;                  /* memory management menu */
MENU *syspmenu=NULLPTR;                 /* system parameters menu */
MENU *rspmenu=NULLPTR;                  /* rsp edit menu */

/*****************************************************************************/
/* main module
/*****************************************************************************/

main(ac,av)
WORD ac;
BYTE **av;
{
        BYTE cmds[CMDSLEN];             /* input area of commands */
        BOOLEAN sysvalid();             /* is the system data valid? */

        fprintf(stderr,edsysver);
```

```
61                  clearit = clearx;       /* yes, do clear screens */
62              else    clearit = "";       /* no, don't do clear screens */
63              init_sysdat();              /* initialize system data */
64              init_edsys();               /* initialize menus */
65              if( ac>1 ) {                /* if any command line arguments */
66                  copyargs(ac-1,av+1,cmds);   /* simplify cmd parser */
67                  verbose = TRUE;         /* turn on wordiness by default */
68                  domenu(cmds,mainmenu);  /* do what they've asked for */
69              }
70              FOREVER {                   /* work with user for a while */
71                      if(doit) {          /* does user want to quit? */
72                          if( sysvalid() )    /* has evrything been defined? */
73                              break;          /* let them if everything okay */
74                          press_return();     /* let them view the message(s) */
75                      }
76                      printf("%s",clearit);   /* clear the screen */
77                      prtmenu(mainmtitle,mainmenu);   /* display user options */
78                      printf(PROMPT);         /* prompt user */
79                      if( gets(cmds)!=cmds ){ /* on end of file */
80                          close(0);           /* standard input */
81                          open("CON:",&READ); /* reopen to the console */
82                          close(1);           /* standard output; assume redirection */
83                          open("CON:",&WRITE);/* reopen to console */
84                          clearit = clearx;   /* do the clear screen routine */
85                          continue;           /* force user to input blank line */
86                      }
87                      if( *cmds==NULL )       /* blank line of input */
88                          continue;           /* don't bother with blank lines here */
89                      domenu(cmds,mainmenu);  /* do what they've asked for */
90                  }                           /*****************************/
91                                              /* we've got user input */
92              make_sysfile();                 /* construct system image/use old one*/
93              gentables();                    /* generate the tables */
94              fixups();                       /* odds and ends on the system image */
95              wrapup();                       /* close files, etc. */
96      }
97
98      /* copyargs: copies args to cbuf for uniform parsing */
99      copyargs(ac,av,cbuf)
100     WORD ac;
101     BYTE **av;
102     BYTE *cbuf;
103     {
104             for( *cbuf=NULL; ac>0; --ac, ++av ) {
105                     strcat(cbuf,*av);
106                     strcat(cbuf," ");
107             }
108     }
109
110     /*************************************************************************/
111     /* init_sysdat: reads SYStem DATa info from:                             */
112     /*       1. SYSFILE, if it exists                                        */
113     /*       2. SYSMOD, o.w. if this is the case, it also checks             */
114     /*               for existence of all other required modules             */
115     /*************************************************************************/
116     VOID init_sysdat()
117     {
118             REG WORD sf;                    /* file descriptor for SysData read */
119             LONG grpseek();                 /* seek the data group */
120             WORD goffs;                     /* loc of group */
```

```
121             EXTERN DLIST *rsps, *dirsearch(); /* directory search fn */
122             BYTE *addmem();                   /* for initializing memory */
123             BYTE *chkmodules(), *xlist[80]; /* list of missing modules */
124
125             printf("GENerate SYStem image for %s\n",VERLABEL);
126             init_sds(sdat_area);              /* initialize program ptrs */
127                             /*******************************************************/
128             if(chkmodules(xlist)==NULLPTR  &&  (sf=BROPEN(SYSMOD)) > 0 ) {
129                 printf("Constructing new %s file\n",SYSFILE);
130                                               /*** procedureize this? for cmd use */
131                 dogensys=TRUE;                /* no SYSFILE, read from modules */
132                 if(grpseek(sf,GTYPDATA)<0)  /* seek start of data group */
133                     CRASH("no data group in SYSDAT module\n");
134                 read(sf,sdat_area,SDLEN);   /* read in the data group */
135                 /**** may have to init arbitrarily some MAX values here ****/
136                 xtrapds = *sd_pul;            /* default number of extra process descriptors */
137                 totqcbs = *sd_qul;            /* default number of qcbs */
138                 totopen = 0x40;               /* arbitrary total open files & locked recs */
139                 qbuflen = *sd_qmalen;         /* init'ed to bytes */
140                 rsps = dirsearch("*.RSP");  /* init RSP list */
141                 addmem("400,6000,400");       /* start out with this memory allocation */
142                 close(sf);                    /* don't need it for a while */
143     #ifdef CCPM
144                 sf=BROPEN(XIOSMOD);           /* let's go read the XIOS info */
145                 if( grpseek(sf,GTYPDATA) <= 0 ) /* if there is not a data group */
146                     grpseek(sf,GTYPCODE);     /* use the code group instead */
147                 goffs = lseek( sf,0L,1 );     /* where did we end up? */
148                 initxios_info(sf,goffs);      /* grab info from there */
149                 close(sf);                    /* don't need it for a while */
150     #endif
151             } else if( FALSE  &&              /****** temporarily disable this option!!! ***/
152                     (sf=BROPEN(SYSFILE)) >= 0 ){ /* does SYSFILE exist? *******/
153                 printf("Editting %s file\n",SYSFILE);
154                 dogensys = FALSE;             /* tell them not to gen a new system */
155                 grpseek(sf,GTYPDATA);         /* get data module in OS */
156                 read(sf,sdat_area,SDLEN);   /* read in the info */
157                 /**** may have to trace down some lists for MAX values ****/
158                 close(sf);                    /* don't need it for a while */
159             } else {        /*******************************************************/
160                 if( xlist == NULLPTR ) {
161                     fprintf(stderr,"Can't find %s module\n",SYSMOD);
162                 } else {
163                     fprintf(stderr,"Can't find these modules:\n%s\n",xlist);
164                 }
165                 fprintf(stderr,"Please find the correct modules\n");
166                 exit(1);
167             }                                 /**** end module check ****/
168             if( VERSION != *sd_mpmvn ){     /* a little version number checking */
169                 USERR("%s works on OS version %x\n",edsysver,VERSION);
170                 USERR("Sys Data area found was from OS version %x\n",*sd_mpmvn);
171                 USERR("Please find correct .SYS or %s files\n",MODEXT);
172                 exit(1);
173             }
174     }
175
176
177     /*****************************************************************************/
178     #define RDAC 4                            /* Read access mode for "access" */
179         BYTE *
180     chkmodules(badbuf)                        /* place to put missing module names */
```

```
181     BYTE *badbuf;
182     {
183
184             *badbuf=0;
185             if( access(SYSMOD,RDAC) ) {strcat(badbuf,SYSMOD);strcat(badbuf," ");}
186             if( access(SUPMOD,RDAC) ) {strcat(badbuf,SUPMOD);strcat(badbuf," ");}
187             if( access(RTMMOD,RDAC) ) {strcat(badbuf,RTMMOD);strcat(badbuf," ");}
188             if( access(MEMMOD,RDAC) ) {strcat(badbuf,MEMMOD);strcat(badbuf," ");}
189             if( access(CIOMOD,RDAC) ) {strcat(badbuf,CIOMOD);strcat(badbuf," ");}
190             if( access(BDOSMOD,RDAC) ) {strcat(badbuf,BDOSMOD);strcat(badbuf," ");}
191     #ifdef NET
192             if( access(NETMOD,RDAC) ) {strcat(badbuf,NETMOD);strcat(badbuf," ");}
193     #endif
194             if( access(XIOSMOD,RDAC) ) {strcat(badbuf,XIOSMOD);strcat(badbuf," ");}
195             if( *badbuf )
196                     return badbuf;
197             else    return NULLPTR;
198     }
199
200
201
202
203     /*************************************************************************/
204     /* init_edsys: initializes program variables and menus                   */
205     /*************************************************************************/
206
207     VOID init_edsys()
208     {
209             MENU *m;                        /* mainmenu pointer */
210             MENU *x;                        /* xiosmenu pointer */
211             MENU *me;                       /* memory partitions menu */
212             MENU *s;                        /* sys params menu */
213             MENU *r;                        /* rsp menu */
214             MENU *bldmenu();                /* func to build the menu list */
215             BYTE *drvmsg;                   /* destination drive message */
216             BYTE *delmsg;                   /* delete old SYSFILE msg */
217             BYTE getdrive();                /* get default drive */
218             BYTE *help();                   /* func to help user */
219             BYTE *doxios();                 /* func to do xios menu */
220             BYTE *dosysp();                 /* func to do sys param menu */
221             BYTE *dorsp();                  /* func to display & handle RSPs */
222             BYTE *dolbl();                  /* func to modify version label */
223             BYTE *domem();                  /* handle memory partitions */
224             BYTE *doxbufs();                /* func to construct bdos buffers */
225             BYTE *addmem();                 /* add mem part */
226             BYTE *delmem();                 /* delete mem part */
227             BYTE *rspinclude();             /* include a bunch of RSPs */
228             BYTE *rspexclude();             /* exclude a bunch of RSPs */
229             BYTE *doexit();                 /* exit the main menu */
230
231             sprintf(mainmtitle,"*** %s %s Main Menu ***",VERLABEL,PROGNAME);
232             destdrv = getdrive();           /* returns the default drive */
233             drvmsg = malloc(50);
234             sprintf(drvmsg,"%s Output To (Destination) Drive",SYSFILE);
235             delmsg = malloc(55);
236             sprintf(delmsg,"Delete (instead of rename) old %s file\n",SYSFILE);
237             m=NULLPTR; x=NULLPTR; me=NULLPTR; s=NULLPTR; r=NULLPTR; /* init menu pointers */
238
239             /***** initialize menus *****/
240     #ifdef CCPM
```

```
241             m=bldmenu(m,MPROC,help,"help","GENCCPM Help");
242             m=bldmenu(m,MBOOL,&verbose,"verbose","More Verbose GENCCPM Messages");
243     #endif
244     #ifdef MPM
245             m=bldmenu(m,MPROC,help,"help","GENMPM Help");
246             m=bldmenu(m,MBOOL,&verbose,"verbose","More Verbose GENMPM Messages");
247     #endif
248             m=bldmenu(m,MDRIV,&destdrv,"destdrive",drvmsg);
249             m=bldmenu(m,MBOOL,&doclean,"deletesys",delmsg);
250
251
252             m=bldmenu(m,MPROC,dosysp,"sysparams","Display/Change System Parameters");
253
254             s=bldmenu(s,MDRIV,sd_srchdisk,"sysdrive","System Drive");
255             s=bldmenu(s,MDRIV,sd_tempdisk,"tmpdrive","Temporary File Drive");
256             s=bldmenu(s,MBOOL,sd_dayfile,"cmdlogging","Command Day/File Logging at Console");
257             s=bldmenu(s,MBOOL,sd_cmode,"compatmode","CP/M FCB Compatibility Mode");
258             s=bldmenu(s,MWORD,sd_mmp,"memmax","Maximum Memory per Process (paragraphs)");
259             s=bldmenu(s,MBYTE,sd_popen_max,"openmax","Open Files per Process Maximum");
260             s=bldmenu(s,MBYTE,sd_plock_max,"lockmax","Locked Records per Process Maximum\n");
261
262             s=bldmenu(s,MWORD,sd_mpmseg,"osstart","Starting Paragraph of Operating System");
263             s=bldmenu(s,MWORD,&totopen,"nopenfiles","Number of Open File and Locked Record Entries");
264             s=bldmenu(s,MBYTE,&xtrapds,"npdescs","Number of Process Descriptors");
265             s=bldmenu(s,MBYTE,&totqcbs,"nqcbs","Number of Queue Control Blocks");
266             s=bldmenu(s,MWORD,&qbuflen,"qbufsize","Queue Buffer Total Size in bytes");
267
268
269             m=bldmenu(m,MPROC,domem,"memory","Display/Change Memory Allocation Partitions");
270             me=bldmenu(me,MPROC,addmem,"add","ADD Memory Partition(s)");
271             me=bldmenu(me,MPROC,delmem,"delete","DELETE Memory Partition(s)");
272
273
274     #ifdef MPM
275             m=bldmenu(m,MPROC,doxios,"xios","Display/Change XIOS Dependent Values");
276
277             x=bldmenu(x,MBYTE,sd_ncns,"nconsoles","Number of System Consoles");
278             x=bldmenu(x,MBYTE,sd_nlst,"nprinters","Number of System Printers");
279             x=bldmenu(x,MBYTE,sd_nccb,"nccbs","Total Character Control Blocks");
280             x=bldmenu(x,MBYTE,sd_nflags,"nflags","Number of Flags");
281             x=bldmenu(x,MBYTE,sd_tickspsec,"nticks","Number of Ticks Per Second");
282     #endif
283     #ifdef CCPM
284             m=bldmenu(m,MPROC,doxbufs,"diskbuffers","Display/Change Disk Buffer Allocation");
285     #endif
286
287             if( dogensys )
288                 m=bldmenu(m,MPROC,dolbl,"oslabel","Display/Change Operating System Label");
289
290             m=bldmenu(m,MPROC,dorsp,"rsps","Display/Change RSP list\n");
291             r=bldmenu(r,MPROC,rspinclude,"include","Include RSPs");
292             r=bldmenu(r,MPROC,rspexclude,"exclude","Exclude RSPs");
293
294
295             m=bldmenu(m,MPROC,doexit,"gensys","I'm finished changing things, go GEN a SYStem\n");
296
297
298             mainmenu=m;                     /* assign now to longer name */
299             xiosmenu=x;                     /* assign to longer name */
300             memmenu=me;                     /*  yah yah */
```

```
301             syspmenu=s;                     /*   "   "  */
302             rspmenu=r;                      /*   "   "  */
303     }
304
305
306         BYTE *
307     doexit()                                /* this is really a boolean... */
308     {
309             doit = TRUE;
310             return NULLPTR;
311     }
312
313     /*************************************************************************/
314     /* sysvalid: validates system data.
315     /*      also calculates certain SYSDAT variables, in the course of validation.
316     /*************************************************************************/
317     #define BYTE_ADJUST(xx) if( xx > 0xFF ) xx = 0xFF
318     #define ALIGN(xx) (((xx)+0xF) & 0xFFF0)
319
320         BOOLEAN
321     sysvalid()
322     {
323             REG BOOLEAN isv;                /* is valid? */
324     #ifdef CCPM
325             BOOLEAN xiosvalid();            /* check xios info */
326     #endif
327             LOCAL WORD nmparts;             /* number of memory partitions */
328
329             isv = TRUE;                     /* innocent until proven guilty */
330             if( totopen < *sd_popen_max ) {
331                     USERR("'nopenfiles' is less than 'openmax'. Please adjust.\n");
332                     isv = FALSE;
333             }
334             if( totopen < *sd_plock_max ) {
335                     USERR("'nopenfiles' is less than 'lockmax'. Please adjust.\n");
336                     isv = FALSE;
337             }
338             if( (nmparts=cntmlist(memroot)) <= 0 ) {
339                     USERR("Memory Partitions need to be adjusted\n");
340                     isv = FALSE;
341             }
342     #ifdef CCPM
343             if( !xiosvalid() ) {            /* xios info needs work? */
344                     USERR("Disk Performance Buffers need to be adjusted\n");
345                     isv=FALSE;
346             }
347     #endif
348             if( !isv )
349                     USERR("Please correct the System Parameters\n");
350             return isv;
351     }
352
353
354
355     /*************************************************************************/
356     /* make_sysfile: responsible for preparing SYSfile image for table generation
357     /* and
358     /*      if( dousesys )
359     /*              doctors up SYSFILE for appropriate table generation and fixups
360     /*      else    constructs a SYSfile from assembled modules
```

```
361   /*************************************************************************/
362   VOID make_sysfile()
363   {
364           BYTE fbuf[20];                  /* file name buffer */
365
366           PUTDRV(fbuf,NEWSYS);
367           unlink(fbuf);                   /* this is the file we'll be using */
368           if( (fns=BWCREAT(fbuf))<0 ) {   /* try to open for output */
369               USERR("can't open new .SYS file (%s)\n",fbuf);
370               exit(1);
371           }
372           if( dogensys )
373                   gensys();               /* make a new SYS file from modules */
374           else    edsys();                /* edit the old SYS file */
375   }
376
377
378   /*************************************************************************/
379   /* gensys: generates a new SYS file image from modules
380   /*************************************************************************/
381       VOID
382   gensys()
383   {
384           WORD fxm;                       /* file descriptor for xios mod */
385           BYTE fbuf[20];
386
387           if( doclean ) {                 /* clean up the SYSFILE first? */
388               PUTDRV(fbuf,SYSFILE);
389               unlink(fbuf);               /* delete if there... */
390           }
391           if(verbose) printf("Generating new SYS file\n");
392           FCHECK(fns);                    /* File error check */
393           write(fns,cmdhdr,SECSIZ);       /* write out a (dummy) command hdr */
394           genfix();                       /* remember the sysdat & sup fixups */
395           clsup = xfergrp(SUPMOD,GTYPCODE);
396           clos_label = writelbl();        /* write the OS label (returns 0 if none exists) */
397           if( clos_label != 0 )
398                   *sd_verptr = clsup<<4;  /* fill in offset of label rel sup code seg */
399           genfix(); clrtm = xfergrp(RTMMOD,GTYPCODE);
400           genfix(); clmem = xfergrp(MEMMOD,GTYPCODE);
401           genfix(); clcio = xfergrp(CIOMOD,GTYPCODE);
402           genfix(); clbdos= xfergrp(BDOSMOD,GTYPCODE);
403   #ifdef NET
404           genfix(); clnet = xfergrp(NETMOD,GTYPCODE);
405   #else
406           clnet = 0;
407   #endif
408           fxm = BROPEN(XIOSMOD);          /* look at XIOS module */
409           if( grpseek(fxm,GTYPDATA) >= 0){        /* is there a data group?*/
410   #ifdef CCPM
411                   USERR("XIOS has separate code and data (small model)");
412                   USERR("This is not supported in this O.S.: use 8080 model.");
413                   exit(1);
414   #else
415                   pure_xios = TRUE;       /* ifso, it's separate code & data */
416                   genfix(); clxios = xfergrp(XIOSMOD,GTYPCODE);
417   #endif
418           } else {
419                   pure_xios = FALSE;
420                   clxios = 0;
```

```
421             }
422             cltotal = clsup+clos_label+clrtm+clmem+clcio+clbdos+clnet+clxios;
423             /*** finished writing the code portion of NEWSYS ***/
424
425             fldstart = lseek(fns,0L,1);     /* where in file SYSDAT gets written */
426             dsstart = *sd_mpmseg + cltotal; /* where in memory SYSDAT gets loaded */
427             dlsysdat = xfergrp(SYSMOD,GTYPDATA);    /* system data area, fixed later */
428
429             if( pure_xios )
430                     dlxios = xferpart_grp(fxm,GTYPDATA,dlsysdat);
431             else { genfix(); dlxios = xferpart_grp(fxm,GTYPCODE,dlsysdat); }
432             close(fxm);                     /* don't need this file descriptor */
433             dltotal = dlsysdat+dlxios;      /* this will change later */
434             /*** finished writing STATIC data portion of NEWSYS ***/
435
436             /*** fill in sysdat here ***/
437             sd_supmod->ep_eseg  = sd_supmod->ep_iseg  = *sd_mpmseg;
438             sd_rtmmod->ep_eseg  = sd_rtmmos->ep_iseg  = sd_supmod->ep_eseg+clsup+clos_label;
439             sd_memmod->ep_eseg  = sd_memmod->ep_iseg  = sd_rtmmod->ep_eseg+clrtm;
440             sd_ciomod->ep_eseg  = sd_ciomod->ep_iseg  = sd_memmod->ep_eseg+clmem;
441             sd_bdosmod->ep_eseg = sd_bdosmod->ep_iseg = sd_ciomod->ep_eseg+clcio;
442             sd_xiosmod->ep_eseg = sd_xiosmod->ep_iseg = sd_bdosmod->ep_eseg+clbdos;
443     #ifdef NET
444             sd_netmod->ep_eseg  = sd_netmod->ep_iseg  = sd_xiosmod->ep_eseg+clxios;
445     #endif
446             if( pure_xios )                 /* compute xios offsets */
447                     sd_xiosmod->ep_ioff = 0;        /* separate code */
448             else    sd_xiosmod->ep_ioff = dlsysdat<<4;      /* mixed code & data, after sysdat */
449             sd_xiosmod->ep_eoff = sd_xiosmod->ep_ioff + 3;
450     }
451
452     /*****************************************************************************/
453     genfix()                /* subroutine to remember SYSDAT & Super fixups */
454     {
455             LONG fa;
456
457             fa = lseek(fns,0L,1);           /* where are we in this file? */
458             fixlater(fa+6,F_PUT,&dsstart);  /* fix the SYSDAT address */
459             fixlater(fa+8,F_PUT,&(sd_supmod->ep_eoff));     /* Supervisor entry offset */
460             fixlater(fa+10,F_PUT,&(sd_supmod->ep_eseg));    /*      "        "   segment */
461     }
462
463
464     /*****************************************************************************/
465     /* edsys: edit the old SYS file image.
466     /*      - copies old sys file into new
467     /*      - if( donew_xios ) will patch in a new XIOS from module
468     /*****************************************************************************/
469         VOID
470     edsys()
471     {
472             if(verbose) printf("Editting old SYS file\n");
473             write(fns,cmdhdr,SECSIZ);       /* leave room for command hdr */
474             cltotal = xfergrp(SYSFILE,GTYPCODE);
475             dltotal = xfergrp(SYSFILE,GTYPDATA);
476             if( donew_xios ) {
477                 ; /* need to read 'xt' from XIOS module */
478             } else {
479     #ifdef CCPM
480                 ; /* need to init 'xt' from sysdat */
```

```
481     #endif
482             )
483     )
484
485
486
487     /************************************************************************/
488     /* fixups: inserts updated SYSDAT page into NEWSYS file, and does some  */
489     /*       address fixups                                                 */
490     /************************************************************************/
491     VOID fixups()
492     {
493             REG WORD ii;                    /* counter */
494             GROUP *cg, *dg;                 /* group ptrs */
495             UWORD genmfl();                 /* (re)Gen Memory Free List */
496
497             if(verbose) printf("Doing fixups\n");
498             FCHECK(fns);                    /* File error check */
499                                             /*** fixup the memory list ***/
500             if(verbose) {
501                 printf("SYS image load map:\n");
502                 printf("         Code starts at %4.4x\n",*sd_mpmseg);
503                 printf("         Data starts at %4.4x\n",dsstart);
504                 printf("       Tables start at %4.4x\n",(dsotables>>4)+dsstart);
505                 printf("         RSPs start at %4.4x\n",(dsotables>>4)+dsstart+dltables);
506                 printf(" XIOS buffers start at %4.4x\n",rsvd_seg);
507                 printf("           End of OS at %4.4x\n",*sd_endseg);
508             }
509
510             if( trimlist(*sd_mpmseg,*sd_endseg) ) { /* mem part list trimmed? */
511                 printf("Trimming memory partitions. New list:");
512                 dspmlist(memroot);
513                 lseek( fns, locmfl, 0 );
514                 ii = (dlsysdat+dlxios)<<4;
515                 genmfl(ii);                 /*****/
516             }
517     #ifdef CCPM
518                                             /*** fixup XIOS info ***/
519             fixupxios_info();               /* fixes the buffer info (DPH info) */
520     #endif
521                                             /*** fixup the command header ***/
522             for( ii=0; ii<SECSIZ; ii++ )    /* zero out command header */
523                 cmdhdr[ii] = '\0';
524             cg = cmdhdr;                    /* code group first */
525             cg->gtype = GTYPCODE;           /* init all code vars */
526             cg->glen = cltotal;             /* total length of code groups */
527             cg->gmin = cltotal;             /* min length of code group */
528             cg->gabase = *sd_mpmseg;        /* loads at absolute address */
529             dg = cmdhdr + sizeof(*cg);      /* data group next */
530             dg->gtype = GTYPDATA;           /* init all data vars */
531             dg->glen = dltotal;             /* total length of data group+tables */
532             dg->gmin = dltotal;             /* min length of data group */
533             dg->gabase = cg->gabase+cltotal;/* abs addr of start of system data */
534             lseek( fns, 0L, 0 );            /* seek the beginning of NEWSYS */
535             write(fns,cmdhdr,SECSIZ);       /* write out the command header */
536
537                                             /*** fixup the sysdat page ***/
538             lseek(fns,fldstart,0);          /* seek there in file */
539             write(fns,sdat_area,SDLEN);     /* put it into the file */
540
```

```
541                                     /*** take care of the little fixes ***/
542             fixfile();              /* fixes from the list */
543     }
544
545     /*************************************************************************/
546     /* wrapup: closes files                                                  */
547     /*************************************************************************/
548     VOID wrapup()
549     {
550             BYTE buf1[20], buf2[20];
551
552             if(verbose) printf("Wrapping up\n");
553             FCHECK(fns);            /* File error check */
554             lseek(fns,0L,2);        /* flush NEWSYS buffers? */
555             if(strlen(edsysver) != write(fns,edsysver,strlen(edsysver))) {
556                 USERR("WRITE FAILURE - the disk may be too full\n");
557                 exit(1);
558             }
559             close(fns);             /* all finished with NEWSYS */
560             PUTDRV(buf1,OLDSYS); unlink(buf1);
561             PUTDRV(buf2,SYSFILE); rename(buf2,buf1);
562             PUTDRV(buf1,NEWSYS); rename(buf1,buf2);
563     }
564
565
566
567     /*************************************************************************/
568     /*************************************************************************/
569     /* edszzz.c: the rest of edsys.c, broken up for shorter compiles */
570     /*************************************************************************/
571     /*************************************************************************/
572     #ifndef MAINMODULE
573     #include <genccpm.h>
574     #endif
575
576
577     /*************************************************************************/
578     /* init_sds: points the system data pointers to the right place          */
579     /*************************************************************************/
580         VOID
581     init_sds(sysdat)
582         BYTE *sysdat;               /* ptr to system data buffer */
583     {
584             sd_supmod      = sysdat + 0x00; /* (4) */
585             sd_rtmmod      = sysdat + 0x08; /* (4) */
586             sd_memmod      = sysdat + 0x10; /* (4) */
587             sd_ciomod      = sysdat + 0x18; /* (4) */
588             sd_bdosmod     = sysdat + 0x20; /* (4) */
589             sd_xiosmod     = sysdat + 0x28; /* (4) */
590             sd_netmod      = sysdat + 0x30; /* (4) */
591             sd_mpmseg      = sysdat + 0x40;
592             sd_rspseg      = sysdat + 0x42;
593             sd_endseg      = sysdat + 0x44;
594             sd_module_map  = sysdat + 0x46;
595             sd_ncns        = sysdat + 0x47;
596             sd_nlst        = sysdat + 0x48;
597             sd_nccb        = sysdat + 0x49;
598             sd_nflags      = sysdat + 0x4A;
599             sd_srchdisk    = sysdat + 0x4B;
600             sd_mmo         = sysdat + 0x4C;
```

```
601             sd_nslaves      = sysdat + 0x4E;
602             sd_dayfile      = sysdat + 0x4F;
603             sd_tempdisk     = sysdat + 0x50;
604             sd_tickspsec    = sysdat + 0x51;
605             sd_lul          = sysdat + 0x52;
606             sd_ccb          = sysdat + 0x54;
607             sd_flags        = sysdat + 0x56;
608             sd_mdul         = sysdat + 0x58;
609             sd_nxmds        = sysdat + 0x58;
610             sd_mfl          = sysdat + 0x5A;
611             sd_nmparts      = sysdat + 0x5A;
612             sd_pul          = sysdat + 0x5C;
613             sd_nxpd         = sysdat + 0x5C;
614             sd_qul          = sysdat + 0x5E;
615             sd_nqds         = sysdat + 0x5D;
616             sd_qmau         = sysdat + 0x60;   /* (4) */
617             sd_qmastart     = sysdat + 0x62;
618             sd_qmalen       = sysdat + 0x64;
619             sd_verptr       = sysdat + 0x78;
620             sd_vn           = sysdat + 0x7A;
621             sd_mpmvn        = sysdat + 0x7C;
622             sd_ncondev      = sysdat + 0x83;
623             sd_nlstdev      = sysdat + 0x84;
624             sd_nciodev      = sysdat + 0x85;
625     #ifdef MPM
626             sd_plock_max    = sysdat + 0x8A;
627             sd_popen_max    = sysdat + 0x8C;
628     #endif
629     #ifdef CCPM
630             sd_lcb          = sysdat + 0x86;
631             sd_popen_max    = sysdat + 0x8A;
632             sd_plock_max    = sysdat + 0x8B;
633     #endif
634             sd_cmode        = sysdat + 0x90;
635     }
636
637
638     /*************************************************************************/
639     /* doxios: gets miscellaneous XIOS values from user */
640     /*************************************************************************/
641         BYTE *
642     doxios(bb,xmtitle)                      /* submenu for XIOS value mods */
643         BYTE *bb;                           /* value passed in from menu */
644         BYTE *xmtitle;
645     {
646             BYTE cmds[CMDSLEN];             /* space for additional commands */
647             EXTERN MENU *xiosmenu;          /* list of menu items for xios menu */
648             EXTERN BYTE *clearit;
649
650             if( bb != NULLPTR ) {           /* if any "command line" arguments */
651                 domenu(bb,xiosmenu);        /* do what they've asked for */
652             } else FOREVER {                /* work with user for a while */
653                     printf("%s",clearit);   /* clear screen */
654                     prtmenu(xmtitle,xiosmenu);      /* display user options */
655                     printf(PROMPT);         /* prompt user */
656                     if( gets(cmds)!=cmds || /* on end of file */
657                         *cmds==NULL )       /* or blank line of input */
658                         break;              /* exit interaction with user */
659                     domenu(cmds,xiosmenu);  /* do what they've asked for */
660             }                               /*****************************/
```

```
661             return NULLPTR;                 /* standard return... */
662     }
663
664
665
666     /************************************************************************/
667     /* dosysp: gets miscellaneous system paramaters from user */
668     /************************************************************************/
669         BYTE *
670     dosysp(bb,smtitle)                      /* submenu for SYSP value mods */
671         BYTE *bb;                           /* value passed in from menu */
672         BYTE *smtitle;
673     {
674             BYTE cmds[CMDSLEN];             /* space for additional commands */
675             EXTERN MENU *syspmenu;          /* list of menu items for sysp menu */
676             EXTERN BYTE *clearit;
677
678             if( bb != NULLPTR ) {           /* if any "command line" arguments */
679                 domenu(bb,syspmenu);        /* do what they've asked for */
680             } else FOREVER {                /* work with user for a while */
681                     printf("%s",clearit);   /* clear screen */
682                     prtmenu(smtitle,syspmenu);      /* display user options */
683                     printf(PROMPT);         /* prompt user */
684                     if( gets(cmds)!=cmds || /* on end of file */
685                         *cmds==NULL )       /* or blank line of input */
686                         break;              /* exit interaction with user */
687                     domenu(cmds,syspmenu);  /* do what they've asked for */
688             }                               /*****************************/
689             return NULLPTR;                 /* standard return... */
690     }
691
692
693
694     /************************************************************************/
695     /* dorsp: handles rsp editting */
696     /************************************************************************/
697         BYTE *
698     dorsp(parm,xmtitle)                     /* handle RSP list editting */
699         BYTE *parm;                         /* command value parameter */
700         BYTE *xmtitle;
701     {
702             BYTE cmds[CMDSLEN];             /* space for additional commands */
703             EXTERN MENU *rspmenu;           /* list of menu items for rsp menu */
704             EXTERN BYTE *clearit;
705
706             if( parm != NULLPTR ) {         /* if any "command line" arguments */
707                 domenu(parm,rspmenu);       /* do what they've asked for */
708             } else FOREVER {                /* work with user for a while */
709                     printf("%s",clearit);   /* clear screen */
710                     rsp_display(rsps);      /* display what they have to work with */
711                     prtmenu(xmtitle,rspmenu);       /* display user options */
712                     printf(PROMPT);         /* prompt user */
713                     if( gets(cmds)!=cmds || /* on end of file */
714                         *cmds==NULL )       /* or blank line of input */
715                         break;              /* exit interaction with user */
716                     domenu(cmds,rspmenu);   /* do what they've asked for */
717             }                               /*****************************/
718             return NULLPTR;                 /* standard return... */
719     }
720
```

```
721   /*************************************************************************/
722       BYTE *
723   rspinclude(rs)
724       BYTE *rs;
725   {
726           BYTE *bp, *cp, *ep;             /* ptrs to take apart command str */
727           DLIST *dd;
728
729           if( rs==NULLPTR || *rs==NULL )
730               return "You need to say 'include=xx.rsp,yy.rsp,zz.rsp'";
731           for( ep=rs; *ep; ++ep ) ;       /* end of string pointer */
732           for( bp=rs; bp<ep; bp=cp+1 ){   /* process each rsp spec */
733               for( cp=bp; *cp && *cp!=','; ++cp ) ;       /* find eos or comma */
734               if( *cp == ',' ) *cp = '\0'; /* null terminate the spec */
735               if( (dd=dirsearch(bp)) == NULLPTR )
736                   return "Can't find RSP to include";
737               rspjoin(dd);
738           }
739           return NULLPTR;
740   }
741
742   rspjoin(dd)
743       DLIST *dd;
744   {
745           DLIST *rs;
746
747           if( rsps == NULLPTR )
748               rsps = dd;
749           else {
750               for( rs=rsps; rs->dlnext != NULLPTR; rs=rs->dlnext ) ;
751               rs->dlnext = dd;
752           }
753   }
754
755
756
757   /*************************************************************************/
758       BYTE *
759   rspexclude(rs)
760       BYTE *rs;
761   {
762           BYTE *bp, *cp, *ep;
763           BOOLEAN rspzap();
764
765           if( rs==NULLPTR || *rs==NULL )
766               return "You need to say 'exclude=aa.rsp,bb.rsp,cc.rsp'";
767           for( ep=rs; *ep; ++ep ) ;       /* find eos */
768           for( bp=rs; bp<ep; bp=cp+1 ){   /* process each rsp spec */
769               for( cp=bp; *cp && *cp!=','; ++cp ) ;       /* find eos or comma */
770               if( *cp == ',' ) *cp = '\0'; /* null terminate the spec */
771               if( !rspzap(bp) )
772                   return "Can't find RSP to exclude";
773           }
774           return NULLPTR;
775   }
776
777       BOOLEAN
778   rspzap(nm)
779       BYTE *nm;
780   {
```

```
781             DLIST **qs, *ps, *rs;
782
783             if( *nm == '*' ) {              /* zap entire list? */
784                 for( ps=rsps; ps!=NULLPTR; ps=rs ) {
785                     rs = ps->dlnext;
786                     free(ps);
787                 }
788                 rsps = NULLPTR;
789                 return TRUE;
790             }
791             for( qs= &rsps; *qs!=NULLPTR; qs = &rs->dlnext ) {      /* zap part of list */
792                 rs = *qs;
793                 if( namematch(nm,rs->dlname) ) {
794                     *qs = rs->dlnext;
795                     free(rs);
796                     return TRUE;
797                 }
798             }
799             return FALSE;                   /* thru loop, can't find it */
800     }
801
802     namematch(s1,s2)
803         BYTE *s1, *s2;
804     {
805             for( ; *s1 && *s2; ++s1, ++s2 )
806                 if( toupper(*s1) != toupper(*s2) )
807                     return FALSE;
808             if( *s1 || *s2 )
809                  return FALSE;
810             else return TRUE;
811     }
812
813
814     /*************************************************************************/
815     #define PAGEWIDTH 72
816     rsp_display(rp)
817     DLIST *rp;
818     {
819             WORD ii;
820
821             printf("\nRSPs to be included are:\n");
822             for( ii=0; rp!=NULLPTR; rp = rp->dlnext ) {
823                 if( (ii += 15) > PAGEWIDTH ) {
824                     printf("\n");
825                     ii=15;
826                 }
827                 printf("   %12.12s",rp->dlname);
828             }
829             printf("\n");
830     }
831
832
833     /*************************************************************************/
834     /* grpseek: seeks in MOD file 'fd' to the start of a code/data group, */
835     /*      returns length (in bytes) of that group */
836     /*************************************************************************/
837         LONG
838     grpseek(fde,gt)
839         WORD fde;                           /* file descriptor */
840         WORD gt;                            /* group type to look for */
```

```
901                     break;
902                 yy += grp.glen;             /* count #pgphs in this group to skip */
903             }
904             if( grp.gtype!=gt ) CRASH("module doesn't have group\n");
905             xx = yy << 4;                   /* cvt to bytes */
906             if( lseek(fm,xx,0)<0 ) CRASH("couldn't seek in module\n");
907             xx = grp.glen; xx -= skip;      /* pgphs to xfer */
908             for( yy=xx<<4; yy>0; yy-=ii ){  /* xfer bytes to NEWSYS */
909                 ii = ( yy > BUFSIZ ? BUFSIZ : yy ); /* num bytes per xfer */
910                 if( read(fm,xferbuf,ii) != ii ) CRASH("not enough bytes in module\n");
911                 write(fns,xferbuf,ii);
912             }
913             return (WORD)(xx);              /* convert to pgphs */
914     }
915
916
917
918     /************************************************************************/
919     /* help: prints out a help message */
920     /************************************************************************/
921
922     BYTE *helpm1[] = {
923             "\n\n",
924             "\t\t*** GENCCPM Help Function ***",
925             "\t\t==========================",
926             "      GENCCPM lets you edit and/or generate a system image from",
927             "operating system modules on the default drive.  A detailed",
928             "explanation of each parameter may be found in the Concurrent CP/M-86",
929             "System Guide, Section 2.\n",
930             "      GENCCPM assumes the default values shown within square",
931             "brackets.  All numbers are Hexadecimal.  To change a parameter,",
932             "enter the parameter name followed by '=' and the new value.  Type",
933             "<CR> (a carriage return) to enter the assignment.  You can make",
934             "multiple assignments if you separate them by a space.  No spaces",
935             "are allowed within an assignment.  Example:\n",
936             "CHANGES?  verbose=N sysdrive=A: openmax=1A <CR>\n",
937             "Parameter names may be shortened to the minimum combination of",
938             "letters unique to the currently displayed menu.  Example:\n",
939             "CHANGES?  v=N sysd=A: op=1a <CR>\n",
940             NULLPTR1
941     };
942     BYTE *helpm2[] = {
943             "\n\n",
944             "Sub-menus (the last few options without default values) are accessed",
945             "by typing the sub-menu name followed by <CR>.  You may enter",
946             "multiple sub-menus, in which case each sub-menu will be displayed",
947             "in order.  Example:\n",
948             "CHANGES?  help sysparams rsps <CR>\n",
949             "Enter <CR> alone to exit a menu, or a parameter name, '=' and the",
950             "new value to assign a parameter.  Multiple assignments may be",
951             "entered, as in response to the Main Menu prompt.\n",
952             NULLPTR1
953     };
954
955
956         BYTE *
957     help(tx)                                /* print a helpful message to user */
958         BYTE *tx;
959     {
960             REG BYTE **cpp;         /* pointer to help strings */
```

```
961
962             for( cpp=helpm1; *cpp != NULLPTR; cpp++ )
963                     printf("%s\n",*cpp);
964             press_return();
965             for( cpp=helpm2; *cpp != NULLPTR; cpp++ )
966                     printf("%s\n",*cpp);
967             press_return();
968             return NULLPTR;
969     }
970
971     /*************************************************************************/
972     /* memory list allocation */
973     /*************************************************************************/
974
975
976         BYTE *
977     domem(cmdval,memtitle)
978         BYTE *cmdval;
979         BYTE *memtitle;
980     {
981             WORD cntmlist();                /* does list validation */
982             BOOLEAN okl;                    /* okay list */
983             BYTE cmds[CMDSLEN];             /* place to put input */
984             EXTERN MLIST *memroot;          /* memory list root */
985             EXTERN MENU *memmenu;           /* memory edit menu */
986             EXTERN BYTE *clearit;
987
988             okl = (cntmlist(memroot) > 0);  /* overlap chk & condense */
989             if(cmdval!=NULLPTR) {
990                 domenu(cmdval,memmenu);     /* do the commands */
991                 okl = (cntmlist(memroot)>0);/* check the results */
992             }
993             if( !okl || cmdval==NULLPTR )   /* do we need to interact with user? */
994                 FOREVER {                   /* display, the edit */
995                     printf("%s",clearit);   /* clear the screen (maybe) */
996                     dspmlist(memroot);      /* here's what it looks like */
997                     prtmenu(memtitle,memmenu);      /* here's how to edit */
998                     printf(PROMPT);         /* what to do? */
999                     if( gets(cmds)!=cmds || *cmds==NULL ) { /* see if the want to get out */
1000                        if(okl)
1001                            break;
1002                        else {
1003                            printf("Please adjust memory partitions\n");
1004                            press_return();
1005                            continue;
1006                        }
1007                    }
1008                    domenu(cmds,memmenu);   /* edit the memory list */
1009                    okl = (cntmlist(memroot)>0);/* validate the memory list */
1010                }
1011            return NULLPTR;                 /* standard return */
1012    }
1013
1014
1015    /*************************************************************************/
1016        BYTE *
1017    addmem(ms)                              /* add memory partitions */
1018        BYTE *ms;                           /* memory partition spec */
1019    {
1020            REG BYTE *pl;                   /* pointer to last addr spec */
```

```
1021            REG BYTE *ps;                   /* pointer to size spec */
1022            UWORD f, l, s;                  /* values of spec */
1023            MLIST *intomlist(), *mnew;      /* where to put them */
1024            EXTERN MLIST *memroot;          /* memory partitions list */
1025            BYTE *badspec = "Add memory partition spec should look like:\n\tadd=first,last,size";
1026            BYTE *badval = "Spec: add=first,last,size\n\twhere last>first, size>80";
1027            BYTE *index();
1028
1029            if( (pl=index(ms,','))==0)      /* look for ptr to last addr */
1030                    return badspec;
1031            else    *pl++ = NULL;           /* terminate first addr spec */
1032            if( (ps=index(pl,','))==0)      /* look for ptr to size */
1033                    return badspec;
1034            else    *ps++ = NULL;
1035            f=atoih(ms);                    /* convert first addr spec */
1036            s=atoih(ps);                    /* convert size spec */
1037            if( *pl=='+' )                  /* last addr relative to first? */
1038                    l=atoih(pl+1)+f;        /* yes, so calculate */
1039            else if( *pl==NULL  ||  *pl=='*' ) {
1040                if( *pl=='*' )              /* last addr multiple of size? */
1041                     l=atoih(pl+1);         /* spec multiple */
1042                else l=1;                   /* default multiple=1 */
1043                l = f+l*s;                  /* s in paragraphs */
1044            } else  l=atoih(pl);            /* no, it's absolute address */
1045            if( l<=f  ||  s==0 )            /* 1st validation check */
1046                    return badval;
1047            if( s<MINKMEM )                 /* 2nd validation check */
1048                    return "Memory partition must be at least 80 paragraphs";
1049            mnew = (MLIST *)malloc(sizeof(*mnew));  /* make room */
1050            mnew->mlfirst = f;
1051            mnew->mllast = l;
1052            mnew->mlsize = s;
1053            memroot=intomlist(memroot,mnew);        /* sorted insertion into list */
1054            if( s > l-f )                   /* check: size_partition > mem_region */
1055                    return "Warning: partition size larger than memory region";
1056            return NULLPTR;                 /* everything okay */
1057    }
1058
1059
1060    /*************************************************************************/
1061        BYTE *
1062    delmem(ms)                              /* delete memory partition */
1063        BYTE *ms;
1064    {
1065            REG WORD df, dl;                /* delete first, last */
1066            MLIST **mm, **mn, *mo, *mp, *mq;        /* necessary list ptrs */
1067            EXTERN MLIST *memroot;          /* memory partitions list */
1068            BYTE *pl;
1069            WORD i;
1070            BYTE *badspec="To delete a memory partition, say\n\tdelete=1  or  delete=1-3\n";
1071
1072            if( *ms == '*' ) {              /* delete all? */
1073                for( mp=memroot; mp!=NULLPTR; mp=mq ) {
1074                    mq = mp->mlnext;        /* save the pointer before we free it */
1075                    free(mp);
1076                }
1077                memroot = NULLPTR;
1078                return NULLPTR;             /* everything okay */
1079            }
1080            if( (pl=index(ms,'-')) == 0 )
```

```
1081                pl=ms;                  /* point last to first */
1082            else    *pl++ = NULL;       /* else point to after "dash" */
1083            df=atoi(ms);                /* decimal conversion */
1084            dl=atoi(pl);
1085            for( i=1, mm=&memroot, mn=NULLPTR, mo=memroot;
1086                i<=dl && mo!=NULLPTR;
1087                ++i ) {
1088                    if( i==df ) mn=mm;      /* found first, pt to ptr to */
1089                    mm = &mo->mlnext;       /* ptr to ptr */
1090                    mo = mo->mlnext;
1091            }
1092            if( mn==NULLPTR || i<=dl )      /* check range of deletions */
1093                return badspec;
1094            mp = *mn;                       /* save ptr to deleted list */
1095            *mn = mo;                       /* cut them suckers out of the list */
1096            for( ; mp!=mo; mp=mq ) {
1097                mq = mp->mlnext;
1098                free(mp);
1099            }
1100            return NULLPTR;                 /* everything ok */
1101    }
1102
1103
1104
1105    /*************************************************************************/
1106    dspmlist(mroot)
1107        MLIST *mroot;
1108    {
1109            MLIST *mn, *mo;
1110            REG WORD i, nump;
1111
1112            printf("\n\n");
1113            printf("          Addresses          Partitions    (in paragraphs)\n");
1114            printf(" #      Start    Last       Size      Qty \n");
1115            for( i=1, mn=mroot; mn!=NULLPTR; ++i, mn=mo ) {
1116                nump = (mn->mllast-mn->mlfirst) / mn->mlsize;
1117                printf("%2.2d.     %4.4xh    %4.4xh      %4.4xh   %4.4xh ",
1118                        i,mn->mlfirst,mn->mllast,mn->mlsize,nump);
1119                if( (mo=mn->mlnext) != NULLPTR ) {  /* do some checking */
1120                    if( mn->mllast > mo->mlfirst )
1121                        printf("**overlaps** ");
1122                    if( mn->mlsize > mn->mllast-mn->mlfirst )
1123                        printf("**partition too big** ");
1124                }
1125                printf("\n");
1126            }
1127    }
1128
1129
1130    /*************************************************************************/
1131        WORD
1132    cntmlist(mroot)                 /* check for overlaps, condense if possible */
1133        MLIST *mroot;               /* returns number mem partitions, 0 if bad list */
1134    {
1135            MLIST *mn, *mo;                 /* ptr within list */
1136            REG BOOLEAN olap;               /* partitions overlap or bad size */
1137            REG WORD nump;                  /* number of partitions this block */
1138            REG WORD totnump;               /* keep the count going */
1139
1140            totnump = 0;                    /* total number partitions */
```

```
1141            olap = FALSE;                   /* Innocent until proven guilty */
1142            for( mn=mroot; mn!=NULLPTR && (mo=mn->mlnext)!=NULLPTR; ) {
1143                if( mn->mlsize==mo->mlsize ){       /* partitions same size? */
1144                  if( mn->mlfirst==mo->mlfirst &&   /* same partitions? */
1145                      mn->mllast==mo->mllast ) {
1146                          mn->mlnext=mo->mlnext;
1147                          free(mo);         /* mo is redundant, zap it */
1148                          continue;
1149                  } else
1150                  if( mn->mllast==mo->mlfirst  ||   /* adjacent partition? */
1151                     mn->mllast==(mo->mlfirst-1) ) {
1152                          mn->mllast=mo->mllast;  /* make 1st partition bigger */
1153                          mn->mlnext=mo->mlnext;  /* zap out mo from list */
1154                          free(mo);         /* we don't need it any more */
1155                          continue;
1156                  }
1157                }
1158                if( mn->mllast > mo->mlfirst )
1159                    olap=TRUE;              /* partitions overlap */
1160                nump = (mn->mllast-mn->mlfirst) / mn->mlsize;
1161                if( nump<=0 )
1162                    olap=TRUE;              /* size > partition */
1163                else totnump += nump;       /* incr this count */
1164                mn=mo;                      /* continue through loop */
1165            }
1166            if( mn!=NULLPTR ) {             /* catch the last in the loop */
1167                nump = (mn->mllast-mn->mlfirst) / mn->mlsize;
1168                if( nump<=0 )
1169                    olap=TRUE;              /* size > partition */
1170                else totnump += nump;       /* incr this count */
1171            }
1172            if( olap )
1173                 return 0;                  /* boo boo, return 0 */
1174            else return totnump;            /* return num partitions */
1175    }
1176
1177
1178
1179    /*************************************************************************/
1180        MLIST *
1181    intomlist(mroot,mi)             /* insert mi into list mroot sorted */
1182        MLIST *mroot;
1183        MLIST *mi;
1184    {
1185            MLIST **mm;                     /* ptr to ptr to list */
1186            MLIST *mn;                      /* ptr to list */
1187
1188            for( mm=&mroot, mn=mroot;
1189                mn != NULLPTR  &&  mi->mlfirst > mn->mlfirst; ){
1190                    mm = &mn->mlnext;
1191                    mn = mn->mlnext;
1192            }
1193            mi->mlnext = mn;
1194            *mm = mi;
1195            return mroot;
1196    }
1197
1198
1199    /*************************************************************************/
1200    /* trimlist: trims the memory list according to OS bounds */
```

```
1201    /************************************************************************/
1202
1203        BOOLEAN                         /* returns TRUE iff any adjustments */
1204    trimlist(osstart,osend)
1205        WORD osstart;                   /* starting pgph */
1206        WORD osend;                     /* ending pgph */
1207    {
1208            EXTERN MLIST *memroot;
1209            REG MLIST **mm, *mn, *mo;   /* ptrs into memlist */
1210            REG WORD ss, zz;            /* temp vars */
1211            LOCAL WORD adjusts;         /* number of collisions */
1212
1213            adjusts = 0;                /* we haven't adjusted anything yet */
1214            for( mm=&memroot, mn=memroot; mn!=NULLPTR; mn=*mm ) {
1215                if( mn->mlfirst < osstart ) {
1216                    if( mn->mllast >= osstart ) { /** collision **/
1217                        ++adjusts;      /* we're doing some adjusting */
1218                        if( mn->mllast > osend ) {  /* real big mem part? */
1219                            mo = (MLIST *)malloc(sizeof *mo); /* set it up now, adjust it later */
1220                            mo->mlfirst = onbounds(mn->mlfirst,mn->mlsize,osstart);
1221                            mo->mllast = mn->mllast;
1222                            mo->mlsize = mn->mlsize;
1223                            mo->mlnext = mn->mlnext;
1224                            mn->mlnext = mo;
1225                        }
1226                        if( (zz=trimit(mn->mlfirst,osstart-1,mn->mlsize)) > 0 ) {
1227                            mn->mllast = osstart-1;
1228                            mn->mlsize = zz;        /* in case it was trimmed */
1229                        } else {            /* partitions too small, delete it */
1230                            *mm = mn->mlnext;
1231                            free(mn);
1232                            continue;       /* don't increment list ptr */
1233                        }
1234                    }                       /**** end collision ****/
1235
1236                } else if( mn->mlfirst < osend ) { /** collision **/
1237                    ++adjusts;              /* we're doing some adjusting */
1238                    if( mn->mllast <= osend ) {     /* does part cross past os? */
1239                        *mm = mn->mlnext;   /* no, delete it */
1240                        free(mn);
1241                        continue;
1242                    }
1243                    ss = onbounds(mn->mlfirst,mn->mlsize,osend);
1244                    if( (zz=trimit(osend,ss,mn->mlsize)) > 0 ){/* set up a smaller mem part */
1245                        mo = (MLIST *)malloc(sizeof *mo);
1246                        mo->mlfirst = osend+1;
1247                        mo->mllast = ss;
1248                        mo->mlsize = zz;
1249                        mo->mlnext = mn;
1250                        *mm = mo;
1251                        mm = &mo->mlnext;
1252                    }
1253                    if( (zz=trimit(ss,mn->mllast,mn->mlsize)) > 0 ){
1254                        mn->mlfirst = ss;
1255                        mn->mlsize = zz;
1256                    } else {                /* partition too small, delete */
1257                        *mm = mn->mlnext;
1258                        free(mn);
1259                        continue;           /* don't bottom out on this loop */
1260                    }
```

```
1261                }                                   /**** end collision ****/
1262
1263                mm = &mn->mlnext;                   /* set up for next partition check */
1264            }                                       /**** end FOR loop ***/
1265            return adjusts>0;                       /* returns TRUE if we adjusted */
1266    }
1267
1268        WORD                /* returns new size of partition, 0 if not enough room */
1269    trimit(start,last,olds)
1270        WORD start;                                 /* start address of partition */
1271        WORD last;                                  /* end address of partition */
1272        WORD olds;                                  /* old partition size */
1273    {
1274            REG WORD ss;
1275
1276            ss = last-start;                        /* size is in paragraphs */
1277            if( ss< 0 ) return 0;                   /* negative size */
1278            if( ss<MINKMEM ) return 0;              /* sorry, too small */
1279            else if( ss<olds ) return ss-1;         /* needs to be smaller */
1280            else return olds;                       /* fine just the way it is */
1281    }
1282
1283        WORD
1284    onbounds(start,size,nst)
1285        WORD start;                                 /* original starting boundary */
1286        WORD size;                                  /* partition size */
1287        WORD nst;                                   /* new starting address */
1288    {                                               /* returns addr>nst multiple of size from start */
1289            REG WORD rr;
1290
1291            for( rr=start; rr<nst; rr += size )
1292                    ;
1293            return rr;
1294    }
1295
1296
1297
1298
1299    /*****************************************************************************/
1300    /*****************************************************************************/
1301    /* edsgen.c                                                                  */
1302    /* EXPORTS gentable() - the routine which builds all the tables              */
1303    /*****************************************************************************/
1304    /*****************************************************************************/
1305    #ifndef MAINMODULE
1306    #include <genccpm.h>
1307    #endif
1308
1309
1310    MLOCAL UWORD rsp_last;          /* link to previous RSP */
1311    MLOCAL UWORD rsp_seg;           /* where next RSP seg gets written */
1312
1313    #define ALIGN(xx) (((xx)+0xF)&0xFFF0)
1314    #define BYTE_ADJUST(yy) if(yy>0xFF) yy=0xFF
1315
1316
1317    /*****************************************************************************/
1318        UWORD                                       /* return new offset */
1319    segwrite(nbs)                                   /* pad to paragraph boundary */
1320        UWORD nbs;                                  /* old offset */
```

```
1321    {
1322            REG UWORD nps;
1323
1324            if( nbs & 0xF ) {               /* does it need padding? */
1325                    nps = 0x10 - (nbs & 0xF);       /* number pad bytes needed */
1326                    write(fns,"\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0",nps);
1327                    return (nbs+nps);
1328            } else  return nbs;
1329    }
1330
1331        UWORD                               /* return num paragraphs */
1332    padwrite(nbs)                           /* pad to paragraph boundary */
1333        UWORD nbs;                          /* num bytes needing padding */
1334    {
1335            return segwrite(nbs) >> 4;              /* convert to pgphs */
1336    }
1337
1338        UWORD
1339    datwrite(from,size)
1340        BYTE *from;
1341        WORD size;
1342    {
1343            if( write(fns,from,size) != size ) {
1344                fprintf(stderr,"ERROR: Write to new SYS file failed\n");
1345                fprintf(stderr,"Out of room on disk?\n");
1346                exit(1);
1347            }
1348            return size;
1349    }
1350
1351    /*************************************************************************/
1352    MLOCAL LONG oldladdr=0;         /* save previous address for comparison */
1353    #define BIGWORD 0x0000ffffL
1354    chkaddr(uaddr)                  /* check for SYSDAT overflow */
1355        UWORD uaddr;
1356    {
1357            LONG laddr;
1358
1359            laddr = BIGWORD & uaddr;
1360            if( laddr < oldladdr ) {
1361                USERR("ERROR - System Data Area has grown too large\n");
1362                USERR("Try reducing the size of some of the tables\n");
1363                exit(1);
1364            }
1365            oldladdr = laddr;       /* save for wrap around compare */
1366    }
1367            /* this is a kluge: I did it because I couldn't get the LONG cast */
1368            /* to work without doing sign extension... whf 1/20/83 */
1369
1370
1371    /*************************************************************************/
1372    /* EXPORT gentables: generates various tables in system data area,
1373    /*      and handles RSPs
1374    /*************************************************************************/
1375    VOID gentables()
1376    {
1377            LOCAL WORD nmparts;             /* number of memory partitions */
1378            LOCAL WORD nsat_items;          /* number of SAT items in qmau */
1379            LOCAL UWORD damds;              /* offset of beginning of tables */
1380            LOCAL UWORD dadbuf;             /* offset of beginning of dbuf */
```

```
1381            LOCAL LONG daend;               /* sys data offset: end of data area */
1382            LOCAL UWORD doffset;            /* current offset into data area */
1383            LOCAL UWORD lastlink;           /* helps link lists together */
1384            LOCAL WORD count;               /* counts num list items being generated */
1385            LOCAL UWORD ucount;             /* unsigned counter */
1386            LOCAL MLIST *mn;                /* for trapsing down memroot */
1387            LOCAL WORD nump;                /* local number partitions */
1388            LOCAL UWORD saddr;              /* local starting addr */
1389            LOCAL WORD length;              /* local partition length */
1390            BYTE buf[SECSIZ];               /* an area for writing */
1391            LOCAL LONG dmagic;              /* fixup location for xt_alloc info */
1392            JWORD genmfl();                 /* gen a memory free list */
1393            UWORD genxios();                /* go gen the xios disk buffers */
1394            DLIST *rs;                      /* ptr to rsp list */
1395            MD md;                          /* sample memory descriptor */
1396            PD pd;                          /*   "     process descriptor */
1397            QCB qcb;                        /*   "     queue control block */
1398            CCB ccb;                        /*   "     channel control block */
1399            LOCK lock;                      /*   "     lock list item */
1400            FLAG flg;                       /*   "     flag item */
1401            SATITEM sat;                    /*   "     sub allocation table item */
1402
1403            if(verbose) printf("Generating tables\n");
1404            FCHECK(fns);                    /* File error check */
1405
1406            /* pre-calculate certain vars */
1407            bufs_seg = 0;                   /* total buf segs allocated in rsvd_seg */
1408            if( (nmparts=cntmlist(memroot)) <= 0 )
1409                    CRASH("invalid memory list, aborting\n");
1410            totpds = xtrapds + nmparts;     /* 1 pd per memory partition, plus xtras */
1411            BYTE_ADJUST(totpds);
1412            totmds = (3*totpds) + nmparts + (nmparts>>1); /* 3 mds per process, plus */
1413                                            /* 1 per partition, plus extra for */
1414                                            /* memory allocator worst case */
1415            BYTE_ADJUST(totmds);
1416            nsat_items = (1+ (1+2*totqcbs)); /* 1 for header entry plus worst case */
1417
1418            dltotal = dlsysdat+dlxios;      /* keep track of total data length in pgphs */
1419            doffset = dltotal << 4;         /* calculate current data offset */
1420            dsotables = doffset;            /* mark the start of tables */
1421            chkaddr(doffset);               /* init the function */
1422
1423                                            /* MEMORY FREE LIST */
1424            locmfl = lseek(fns,0L,1);       /* remember where we are writing this */
1425            doffset = genmfl(doffset);      /* gen the mem free list */
1426
1427            lastlink = 0;                   /* UNUSED MEMORY DESCRIPTOR LIST */
1428            zfill(&md,sizeof md);           /* init the mem descriptor */
1429            for( count=totmds-nmparts; count>0; --count ) { /* generate rest of mds */
1430                md.md_link = lastlink;
1431                lastlink = doffset;
1432                doffset += datwrite(&md,sizeof(md));
1433            }
1434            *sd_mdul = lastlink;            /* store ptr to mds unused */
1435
1436            lastlink = 0;                   /* UNUSED PROCESS DESCRIPTOR LIST */
1437            zfill(&pd,sizeof(pd));
1438            for( count=totpds; count>0; --count ) {
1439                pd.pd_link = lastlink;
1440                lastlink = doffset;
```

```
1441                doffset += datwrite(&pd,sizeof(pd));
1442            }
1443            *sd_pul = lastlink;             /* store ptr to pds unused */
1444
1445            lastlink = 0;                   /* UNUSED QUEUE CONTROL BLOCKS */
1446            zfill(&qcb,sizeof(qcb));
1447            for( count=totqcbs; count>0; --count ) {
1448                qcb.qc_link = lastlink;
1449                lastlink = doffset;
1450                doffset += datwrite(&qcb,sizeof(qcb));
1451            }
1452            *sd_qul = lastlink;             /* store ptr to unused qcb list */
1453
1454            lastlink = 0;                   /* UNUSED LOCKED ITEMS LIST */
1455            zfill(&lock,sizeof(lock));
1456            for( count=totopen; count>0; --count ) {
1457                lock.lo_link = lastlink;
1458                lastlink = doffset;
1459                doffset += datwrite(&lock,sizeof(lock));
1460            }
1461            *sd_lul = lastlink;             /* store ptr to unused lock items list */
1462
1463    #ifdef MPM
1464            *sd_ccb = doffset;              /* CHANNEL CONTROL BLOCK TABLE */
1465            zfill(&ccb,sizeof(ccb));
1466            ccb.cc_mimic = ccb.cc_msource = 0xFF;
1467            for( count= *sd_nccb; count>0; --count ) {
1468                doffset += datwrite(&ccb,sizeof(ccb));
1469            }
1470    #endif
1471    #ifdef CCPM
1472            *sd_ncns = xt.xt_nvcns;
1473            *sd_ncondev = xt.xt_nccbs;
1474            *sd_ccb = xt.xt_ccb;
1475            *sd_nlst = xt.xt_nlcbs;
1476            *sd_nlstdev = xt.xt_nlcbs;
1477            *sd_lcb = xt.xt_lcb;
1478            *sd_nciodev = *sd_ncondev + *sd_nlstdev;
1479            *sd_tickspsec = xt.xt_ticks_sec;
1480    #endif
1481
1482            *sd_flags = doffset;            /* FLAG TABLE */
1483            flg.fl_status = -1;
1484            flg.fl_pd = -1;
1485            for( count= *sd_nflags; count>0; --count ) {
1486                doffset += datwrite(&flg,sizeof(flg));
1487            }
1488
1489
1490    #ifdef CCPM
1491            doffset = genxios(doffset);     /* go gen the xios buffering info */
1492    #endif
1493            chkaddr(doffset);               /* is table space too big? */
1494
1495            doffset = segwrite(doffset);    /* Q Sub Alloc Tab HEADER */
1496            *sd_qmastart = dsstart + (doffset>>4);  /* fill in seg addr */
1497            daqbuf = doffset + sizeof(sat)*nsat_items;
1498            zfill(&sat,sizeof(sat));
1499            sat.sa_start = --nsat_items;    /* decr before write */
1500            doffset += datwrite(&sat,sizeof(sat));  /* write the header */
```

```
1501                                    /* Q Sub Alloc Tab ENTRIES */
1502            sat.sa_start = daqbuf;
1503            sat.sa_length = qbuflen;        /* byte value */
1504            *sd_qmalen = qbuflen>>4;        /* pgph value */
1505            doffset += datwrite(&sat,sizeof(sat));  /* write the first sat entry */
1506
1507            zfill(&sat,sizeof(sat));
1508            for( count= --nsat_items; count>0; --count ) {
1509                doffset += datwrite(&sat,sizeof(sat));/* write the rest of the sat entries */
1510            }
1511
1512            zfill(buf,SECSIZ);              /* Q BUFFER AREA - may go past 64K */
1513            daend = doffset;                /* abandon doffset */
1514            for( ucount=qbuflen; ucount>0; ucount -= length ) {
1515                length = (ucount>SECSIZ ? SECSIZ : ucount);
1516                daend += datwrite(buf,length);
1517            }
1518            doffset = daend & 0xF;          /* get low nibble */
1519            daend = (daend >> 4) + padwrite(doffset); /* cvt to pgphs */
1520
1521            dltables = daend - dltotal;     /* compute length of tables in pgphs */
1522            dltotal = daend;                /* keep track of total data length */
1523
1524            /*** now fill in the rsps ***/
1525            if(verbose) printf("Appending RSPs to system file\n");
1526            FCHECK(fns);                    /* File error check */
1527            rsp_last = 0;
1528            rsp_seg = dsstart + daend;
1529            for( rs=rsps; rs!=NULLPTR; rs=rs->dlnext ) /* follow the rsp list */
1530                xfer_rsp(rs->dlname);
1531            *sd_rspseg = rsp_last;
1532            dlrsps = rsp_seg - (dsstart + daend);
1533            dltotal += dlrsps;
1534            rsvd_seg = rsp_seg;
1535    #ifdef CCPM
1536            length = xt.xt_alloc;           /* how much memory is XIOS requesting */
1537            xt.xt_alloc = rsvd_seg+bufs_seg; /* this is the place they can use */
1538            dmagic = fldstart;              /* location of SYSDAT in SYS file */
1539            dmagic += XTABALLOC;            /* add in where xt_alloc goes to */
1540            fixlater(dmagic,F_PUT,&(xt.xt_alloc));  /* fix this number later on */
1541            bufs_seg += length;             /* allocate requested space */
1542    #endif
1543            *sd_endseg = rsvd_seg+bufs_seg; /* this is where we reserve to */
1544    }
1545
1546
1547    /*************************************************************************/
1548    /* genmfl: writes out a memory free list, assuming file 'fns' is seeking
1549    /*       correctly into file.
1550    /*************************************************************************/
1551        UWORD
1552    genmfl(doffset)
1553        UWORD doffset;
1554    {
1555            REG MLIST *mn;                  /* ptr into EOSYS mem list */
1556            LOCAL WORD numb;
1557            LOCAL UWORD lastlink, length, saddr; /* counter vars */
1558            MD md;                          /* mem desc struct */
1559
1560            *sd_mfl = doffset;              /* gen this list forward */
```

```
1561            zfill(&md,sizeof(md));          /* init the data structure */
1562            for( mn=memroot; mn!=NULLPTR; mn=mn->mlnext ) {
1563                nump = (mn->mllast - mn->mlfirst) / mn->mlsize;
1564                saddr = mn->mlfirst;        /* use this starting address */
1565                length = md.md_length = mn->mlsize; /* size in pgphs */
1566                for( ; nump>0; --nump ) {   /* for each partition */
1567                    doffset += sizeof(md);  /* incr by sizeof object */
1568                    lastlink = doffset;     /* record address of where this will go */
1569                    if( nump == 1 ) {       /* on this group's last partition */
1570                        length = md.md_length = mn->mllast-saddr; /* stick in rmdr */
1571                        if( mn->mlnext == NULLPTR ) /* on the very last partition */
1572                            lastlink=0;     /* set end of list */
1573                    }
1574                    md.md_link = lastlink;  /* thread the list */
1575                    md.md_start = saddr;    /* set the starting addr */
1576                    write(fns,&md,sizeof(md));      /* put it into the NEWSYS file */
1577                    saddr += length;        /* next md starts here */
1578                }
1579            }                               /* finished with memory partitions */
1580            return doffset;
1581    }
1582
1583
1584    /*****************************************************************************/
1585        VOID
1586    zfill(addr,num)
1587        BYTE *addr;
1588        WORD num;
1589    {
1590            for( ; num>0; --num )
1591                    *addr++ = '\0';
1592    }
1593
1594
1595    /*****************************************************************************/
1596    /*****************************************************************************/
1597    #define RSPHDR struct rsp_header
1598    RSPHDR {
1599            UWORD rs_link;                  /* link to other RSPs */
1600            WORD rs_sdatvar;                /* SYSDAT offset: num copies */
1601            BYTE rs_ncopies;                /* local: num copies */
1602            BYTE rs_space[11];              /* fill up rest of header */
1603    };
1604
1605    RSPHDR *rhdr;                           /* RSP's header */
1606    PD *rpd;                                /* RSP's process descriptor area */
1607    UDA *ruda;                              /* RSP's uda area */
1608    #define RHDRSIZE (sizeof(RSPHDR)+sizeof(PD)+sizeof(UDA))
1609    #define RHDRPGPHS (RHDRSIZE>>4)
1610    MLOCAL BYTE rh[RHDRSIZE];               /* where to read the RSP header */
1611    MLOCAL BYTE rspnbuf[9];                 /* room for copy of pd name */
1612    MLOCAL BYTE *rspname;                   /* name of RSP */
1613
1614
1615    /*****************************************************************************/
1616    xfer_rsp(rspnm)
1617        BYTE *rspnm;                        /* name of rsp */
1618    {
1619            LOCAL WORD fr;                  /* rsp file descriptor */
1620            LOCAL WORD ncs;                 /* number of copies */
```

```
1621
1622            rhdr = rh;                      /* init these vars */
1623            rpd = rh+sizeof(*rhdr);         /* points to process descriptor */
1624            ruda = rh+sizeof(*rhdr)+sizeof(*rpd);   /* points to user data area */
1625            if( (fr=BROPEN(rspnm)) < 0 ) {
1626                printf("can't open RSP %s\n",rspnm);
1627                exit(1);
1628            }
1629            if( grpseek(fr,GTYPDATA) > 0L){ /* is there a data group? */
1630                ncs = gwab_hdr(fr);
1631                if( rpd->pd_mem == 0 )
1632                     xfer_separate_rsp(fr,ncs);
1633                else xfer_pure_rsp(fr,ncs);
1634            } else {
1635                grpseek(fr,GTYPCODE);
1636                ncs = gwab_hdr(fr);
1637                if( rpd->pd_mem == 0 )
1638                     xfer_mixed_rsp(fr,ncs);
1639                else {
1640                    USERR("RSP %s has non zero MEM field and no data group\n",
1641                          rspnm);
1642                }
1643            }
1644            close(fr);
1645    }
1646
1647
1648    /*************************************************************************/
1649        WORD
1650    gwab_hdr(fr)
1651        WORD fr;
1652    {
1653            REG WORD rv;                    /* return val */
1654            EXTERN BYTE sdat_area[];
1655
1656            rspname = NULLPTR;              /* blank out this name */
1657            read(fr,rh,RHDRSIZE);
1658            strncpy(rspnbuf,rpd->pd_name,8);/* store this in case we need it */
1659            if( rhdr->rs_sdatvar != 0 ) {
1660                    rspname = rspnbuf;
1661                    rv = sdat_area[rhdr->rs_sdatvar] - 1;
1662            } else {
1663                if( rhdr->rs_ncopies > 0 )
1664                    rspname = rspnbuf;
1665                rv = rhdr->rs_ncopies;
1666            }
1667            return rv;
1668    }
1669
1670
1671    /*************************************************************************/
1672    xfer_separate_rsp(fr,ncs)               /* (many) data and (many) code segs */
1673        WORD fr;                            /* file desc of RSP file */
1674        WORD ncs;                           /* number of copies */
1675    {
1676            WORD ii;                        /* counter */
1677
1678            for( ii=0; ii<=ncs; ++ii) {
1679                ruda->ud_dsinit = ruda->ud_esinit = ruda->ud_ssinit = rsp_seg;
1680                ruda->ud_csinit = rsp_seg + (grpseek(fr,GTYPDATA)>>4);
```

```
1681                rhdr->rs_ncopies = ii;
1682                if( rspname )               /* do we need to insert a modfied name? */
1683                    putname(rpd->pd_name,rspname,ii);
1684                rhdr->rs_link = rsp_last;   /* link together the list */
1685                rsp_last = rsp_seg;         /* remember this for next time */
1686                write(fns,rh,RHDRSIZE);     /* write out the RSP header */
1687                rsp_seg += RHDRPGPHS + xferpart_grp(fr,GTYPDATA,kHDRPGPHS);
1688                rsp_seg += xferpart_grp(fr,GTYPCODE,0);
1689            }
1690    }
1691
1692
1693    /*****************************************************************************/
1694    xfer_pure_rsp(fr,ncs)                   /* (many) data segs and one code segs */
1695        WORD fr;                            /* file desc of RSP file */
1696        WORD ncs;                           /* number of copies */
1697    {
1698            WORD ii;                        /* counter */
1699
1700            ruda->ud_csinit = rsp_seg + (1+ncs) * (grpseek(fr,GTYPDATA) >> 4);
1701            for( ii=0; ii<=ncs; ++ii) {
1702                ruda->ud_dsinit = ruda->ud_esinit = ruda->ud_ssinit = rsp_seg;
1703                rhdr->rs_ncopies = ii;
1704                if( rspname )               /* do we need to insert a modfied name? */
1705                    putname(rpd->pd_name,rspname,ii);
1706                rpd->pd_mem = 8;            /* magic ptr to md in rsphdr */
1707                rhdr->rs_link = rsp_last;   /* link together the list */
1708                rsp_last = rsp_seg;         /* remember this for next time */
1709                write(fns,rh,RHDRSIZE);     /* write out the RSP header */
1710                rsp_seg += RHDRPGPHS + xferpart_grp(fr,GTYPDATA,RHDRPGPHS);
1711            }
1712            rsp_seg += xferpart_grp(fr,GTYPCODE,0);
1713    }
1714
1715
1716    /*****************************************************************************/
1717    xfer_mixed_rsp(fr,ncs)                  /* (many) mixed code/data segs */
1718        WORD fr;                            /* file desc of RSP file */
1719        WORD ncs;                           /* number of copies */
1720    {
1721            WORD ii;                        /* counter */
1722
1723            for( ii=0; ii<=ncs; ++ii) {
1724                ruda->ud_csinit = ruda->ud_dsinit = ruda->ud_esinit =
1725                    ruda->ud_ssinit = rsp_seg;
1726                rhdr->rs_ncopies = ii;
1727                if( rspname )               /* do we need to insert a modfied name? */
1728                    putname(rpd->pd_name,rspname,ii);
1729                rhdr->rs_link = rsp_last;   /* link together the list */
1730                rsp_last = rsp_seg;         /* remember this for next time */
1731                write(fns,rh,RHDRSIZE);     /* write out the RSP header */
1732                rsp_seg += RHDRPGPHS + xferpart_grp(fr,GTYPCODE,RHDRPGPHS);
1733            }
1734    }
1735
1736
1737    /*****************************************************************************/
1738    putname(to,from,num)
1739        BYTE *to;
1740        BYTE *from;
```

```
1741        WORD num;
1742    {
1743            BYTE buf[15];
1744            BYTE *bp;
1745            WORD jj;
1746
1747            for( bp=buf, jj=0; *from  &&  *from != ' '  &&  jj<8; ++jj )
1748                    *bp++ = *from++;
1749            if( jj>6 )
1750                    bp = buf+7;
1751            sprintf(bp,"%x      ",num);
1752            for( bp=buf, jj=0; jj<8; jj++ )
1753                    *to++ = *bp++;
1754    }
1755
1756
1757    /*************************************************************************/
1758    /* dolbl: modify operating system version label */
1759    /*************************************************************************/
1760    #define MAXLBL 80
1761
1762        BYTE *
1763    dolbl(txt,lbtitle)
1764        BYTE *txt;                          /* cmd val */
1765        BYTE *lbtitle;
1766    {
1767            DLIST *sl, *psl;                /* ptrs to system labels */
1768            BYTE mb[MAXLBL];                /* max size of each line */
1769            EXTERN BYTE *clearit;
1770
1771            sl = psl = syslbls;             /* init ptrs */
1772            printf("%s\n%s\n",clearit,lbtitle);
1773            printf("Current message is:\n");
1774            if( syslbls==NULLPTR )
1775                    printf("<null>\n");
1776            else for( ; sl != NULLPTR; sl = (psl=sl)->dlnext )
1777                    printf("%s\n",sl->dlname);
1778            printf("\nAdd lines to message.  Terminate by entering only RETURN:\n");
1779            FOREVER {
1780                if( gets(mb) != mb  ||  *mb==NULL )
1781                    break;
1782                sl = (DLIST *)malloc(sizeof(*sl));  /* alloc a list item */
1783                sl->dlname = malloc(sizeof(mb));
1784                strcpy(sl->dlname,mb);      /* copy message to new space */
1785                sl->dlnext = NULLPTR;
1786                if( psl==NULLPTR )          /* new list? */
1787                    syslbls = psl = sl;     /* this will be the first line */
1788                else {
1789                    psl->dlnext = sl;
1790                    psl = psl->dlnext;
1791                }
1792            }
1793            return NULLPTR;
1794    }
1795
1796    /*************************************************************************/
1797    /* writelbl: writes the OS label to the NEWSYS file, returning the length
1798    /*      of the label (in pgphs)
1799    /*************************************************************************/
1800        WORD
```

```
1801    _wrlbl(str)                             /* subroutine to writelbl */
1802        BYTE *str;
1803    {
1804            REG WORD slen;
1805
1806            slen=strlen(str);               /* length of string to be written */
1807            write(fns,str,slen);            /* write string to file NEWSYS */
1808            write(fns,"\015\012",2);        /* cr,lf */
1809            return slen+2;
1810    }
1811
1812        WORD
1813    writelbl()                              /* returns num pgphs written */
1814    {
1815            REG WORD cc;                    /* count of chars output */
1816            DLIST *sl;                      /* list ptr */
1817
1818            cc = 0;
1819
1820            cc += _wrlbl("");               /* crlf */
1821            cc += _wrlbl(version);          /* write out the version label */
1822            cc += _wrlbl(copyright);        /* followed by copyright notice */
1823            for( sl=syslbls; sl!=NULLPTR; sl=sl->dlnext )
1824                cc += _wrlbl(sl->dlname);   /* followed by all lines in OS label */
1825            cc += _wrlbl("$");              /* terminate message for printstr fn */
1826            return padwrite(cc);            /* writes enough zeroes to pad to pgph bdry */
1827    }
1828
1829
1830    /*************************************************************************/
1831    /* fixup routines: remembers and posthumously patches the SYS file       */
1832    /*************************************************************************/
1833
1834    FIX *fixroot = NULLPTR;                 /* list of fixes */
1835
1836    fixlater(fa,ft,fp)                      /* records the fix in the list */
1837        LONG fa;                            /* addr within NEWSYS to fix */
1838        WORD ft;                            /* type of fix: F_ADD or F_PUT */
1839        WORD *fp;                           /* ptr to val used in fixing */
1840    {
1841            FIX *fx;
1842
1843            fx = (FIX *)malloc(sizeof *fx); /* alloc a list item */
1844            fx->f_addr = fa;
1845            fx->f_type = ft;
1846            fx->f_ptr = fp;
1847            fx->f_next = fixroot;           /* insert at head of list */
1848            fixroot = fx;
1849    }
1850
1851    fixfile()                               /* does the fixes in the file */
1852    {
1853            FIX *fx;
1854            WORD fval;                      /* the value to stuff */
1855            LONG lastval;
1856
1857            lastval=lseek(fns,0L,2);
1858
1859            for( fx=fixroot; fx!=NULLPTR; fx = fx->f_next ) {
1860                if(fx->f_addr > lastval) {
```

```
1861                    printf("fixfile: %X %x %x\n",fx->f_addr,fx->f_type,fx->f_ptr);
1862                    continue;
1863                }
1864                if( lseek(fns,fx->f_addr,0) < 0L )
1865                    CRASH("fixup seek failure\n");
1866                if( fx->f_type == F_ADD ) { /* F_ADD the value to what's there */
1867                    read(fns,&fval,2);
1868                    fval += *(fx->f_ptr);
1869                    lseek(fns,-2L,1);       /* backup to write */
1870                } else                      /* must be a F_PUT */
1871                    fval = *(fx->f_ptr);
1872                write(fns,&fval,2);
1873            }
1874    }
1875
1876
1877
1878
1879    /*****************************************************************************/
1880    /*****************************************************************************/
1881    /* edsdisk - This module exports the following routines:
1882    /*
1883    /* initxios_info(xf,goff)       'xf' is XIOSMOD file descriptor, 'goff'
1884    /*      is offset of data group in XIOSMOD (including header);
1885    /*      this module inits the buffering info questions.
1886    /* doxbufs(txt,title)   'txt' is a command, 'title' is the prompt for this fn;
1887    /*      this routine is called within the MENU module, and prompts user
1888    /*      for xios info.
1889    /* xiosvalid()  returns a BOOLEAN value indicating whether the xios info
1890    /*      is valid.
1891    /* genxios(doffset)  generates the buffers for the xios, starting at 'doffset'
1892    /*      in the system data area.
1893    /* fixupxios_info()     writes DPH fixups required by genxios
1894    /*****************************************************************************/
1895    /*****************************************************************************/
1896    #ifndef MAINMODULE
1897    #include <genccpm.h>
1898    #endif
1899
1900
1901    /*****************************************************************************/
1902    /* edsdisk.h : defines the data structures used in this module
1903    /*****************************************************************************/
1904
1905    #define ADDR UWORD                      /* Address (offset) within SysDat */
1906    #define SEG  UWORD                      /* Segment address (may be outside SysDat) */
1907
1908    #define DPH struct dph_item
1909    DPH {                                   /* Disk Parameter Header structure */
1910            ADDR dph_xlt;                   /* Translation Table Address */
1911            BYTE dph_space[6];              /* reserved & Media Flag */
1912            ADDR dph_dpb;                   /* Disk Parameter Block Address */
1913            ADDR dph_csv;                   /* CheckSum Vector Address */
1914            ADDR dph_alv;                   /* Allocation Vector Address */
1915            ADDR dph_dirbcb;                /* Dir Buffer Control Block Header Address */
1916            ADDR dph_datbcb;                /* Data Buffer Control Block Header Address */
1917            SEG  dph_hstbl;                 /* Hash Table Segment */
1918    };
1919
1920    #define DPB struct dpb_item
```

```
1921    DPB {                                   /* Disk Parameter Block structure */
1922            WORD dpb_spt;                   /* Sectors Per Track */
1923            BYTE dpb_bsh;                   /* Allocation block Shift Factor */
1924            BYTE dpb_blm;                   /* Allocation block Mask */
1925            BYTE dpb_exm;                   /* Extent Mask */
1926            WORD dpb_dsm;                   /* Disk Storage Maximum */
1927            WORD dpb_drm;                   /* Directory Maximum */
1928            WORD dpb_dav;                   /* Directory Allocation Vector */
1929            WORD dpb_cks;                   /* Checksum Vector Size */
1930            WORD dpb_off;                   /* Track Offset */
1931            WORD dpb_psh;                   /* Physical Record Shift Factor */
1932            WORD dpb_phm;                   /* Physical Record Mask */
1933    };
1934
1935    #define BCBH struct bcbh_item
1936    BCBH {                                  /* Buffer Control Block Header structure */
1937            ADDR bcbh_lr;                   /* BCB List Root */
1938            BYTE bcbh_pm;                   /* BCB Process Max */
1939    };
1940
1941    #define BCB struct bcb_item
1942    BCB {                                   /* Buffer Control Block structure */
1943            BYTE bcb_drv;                   /* Drive */
1944            BYTE bcb_record[3];             /* Record Number */
1945            BYTE bcb_wflg;                  /* Write Pending Flag */
1946            BYTE bcb_seq;                   /* Sequential Access Counter */
1947            WORD bcb_track;                 /* Logical Track Number */
1948            WORD bcb_sector;                /* Logical Translated Sector Number */
1949            UWORD bcb_bufptr;               /* Buffer Offset (for DIRBCB) / Segment (for DATBCB) */
1950            ADDR bcb_link;                  /* Link to next BCB */
1951            WORD bcb_pdadr;                 /* Process Descriptor Address */
1952    };
1953
1954
1955    #define DINFO struct dirbuf_information
1956    DINFO {
1957            BOOLEAN d_modified;             /* DPH was modified? */
1958            BOOLEAN d_hashing;              /* user wants hashing? */
1959            WORD d_ndirbs;                  /* Num dir buffers */
1960            WORD d_sdirbs;                  /* Size dir buffers */
1961            WORD d_pmdirbs;                 /* Process Max Number Dir Buffs */
1962            WORD d_ndatbs;                  /* Num data buffers */
1963            WORD d_sdatbs;                  /* Size data buffers */
1964            WORD d_pmdatbs;                 /* Process Max Number Dir Buffs */
1965            WORD d_shash;                   /* Size of hash table */
1966            WORD d_alvsize;                 /* size of Alloc Vector area (in bytes ) */
1967            WORD d_blksize;                 /* size of disk Allocation Block */
1968            WORD d_csvsize;                 /* size of Checksum Vector area */
1969    };
1970
1971    /* end of edsdisk.h ***********************************************************/
1972
1973
1974
1975    /*****************************************************************************/
1976    WORD csfd;                              /* command file descriptor */
1977    WORD csgoff;                            /* group offset within command file */
1978
1979    cseek(offs)                             /* seek offset within Command File */
1980        ADDR offs;
```

```
1981    {
1982            lseek(csfd,(LONG) offs+csgoff, 0);      /* compensate for cmd header */
1983    }
1984
1985    /*************************************************************************/
1986    /* following is temporary!! change to zero after debugging */
1987    #define ZBYTE (doffset&0xFF)            /* the byte usually passed in */
1988        WORD
1989    bufwrite(size,byt)
1990        WORD size;
1991        BYTE byt;
1992    {
1993            WORD ss;
1994
1995            for( ss=size; ss>0; --ss )
1996                    write(fns,&byt,1);
1997            return size;
1998    }
1999
2000
2001    /*************************************************************************/
2002    /* Module local variables */
2003    #define DPHNUM 16                       /* number of possible disks */
2004    ADDR *dphtab;                           /* table of offsets of Disk Parm Hdrs */
2005    DPH *dh[DPHNUM];                        /* table of Disk Parm Hdrs */
2006    DINFO *di[DPHNUM];                      /* table of info on DPHs */
2007
2008
2009    /* Module local predicates */
2010    #define ASKFOR(aa) ((aa) == 0xFFFF)     /* Is this one we're sposed to ask for? */
2011    #define SHRMSK 0x8000                   /* Is this shared with another drive? */
2012    #define SHARED(bb) ((bb)!=0xFFFF  &&  (bb) & SHRMSK)    /* Is this nd??bs field shared with another disk? */
2013    #define GETSHARED(cc) ((cc) & ~SHRMSK)  /* Get the drive we share with */
2014
2015
2016
2017    /*************************************************************************/
2018    /* EXPORT
2019    /* initxios_info: gets whatever information GENSYS needs from xios
2020    /*************************************************************************/
2021    initxios_info(xf,goff)                  /* get info from xios file */
2022        WORD xf;                            /* xios file descriptor */
2023        WORD goff;                          /* group offset within xios file */
2024    {
2025            REG WORD ii;                    /* counter */
2026            DPB dd;                         /* a place to keep this info */
2027            DPH *dph;
2028            DINFO *dinf;
2029
2030            csfd = xf;                      /* assign to global: set up cseek */
2031            csgoff = goff;                  /* assign to global */
2032            cseek(XTABLOC);                 /* go to where the XIOSTAB info is */
2033            read(xf,&xt,sizeof xt);         /* read in the xios info */
2034            dphtab = xt.xt_dphtab;          /* point to disk parm header table */
2035
2036            for( ii=0; ii<DPHNUM; ++ii ) {  /* for each dph pointer */
2037                if( dphtab[ii] == 0 ) {     /* is there a dph? */
2038                    dh[ii]=NULLPTR;
2039                    di[ii]=NULLPTR;         /* no ticket, no washee */
2040                } else {                    /* aha! let's set up this info */
```

```
2041                    dh[ii] = (DPH *)malloc(sizeof(DPH));
2042                    di[ii] = (DINFO *)malloc(sizeof(DINFO));
2043                    zfill(di[ii],sizeof(DINFO));
2044                    cseek(dphtab[ii]);
2045                    read(xf,dh[ii],sizeof(DPH));
2046                }
2047            }                               /* end of reading dph */
2048            for( ii=0; ii<DPHNUM; ++ii )    /* let's go thru it again */
2049                if( dh[ii] ) {              /* if there's something there */
2050                    dph = dh[ii];
2051                    dinf = di[ii];
2052                    cseek(dph->dph_dpb);    /* point to the disk parm block */
2053                    read(xf,&dd,sizeof(DPB));
2054                    dinf->d_blksize = 128 << dd.dpb_psh;
2055                    dinf->d_csvsize = 0x7FFF & dd.dpb_cks;
2056                    dinf->d_alvsize = (dd.dpb_dsm/8 + 1) * 2;
2057                    dinf->d_shash = 4 * (1 + dd.dpb_drm);
2058                    dinf->d_ndirbs = -1;    /* ask for these by name! */
2059                    dinf->d_ndatbs = -1;
2060                    dinf->d_hashing = TRUE; /* default to hashing */
2061                }                           /* end of drive info init */
2062    }
2063
2064
2065
2066    /*************************************************************************/
2067    /* EXPORT
2068    /* doxbufs: queries user for BDOS buffering info;
2069    /*      exits only when all data valid
2070    /*************************************************************************/
2071        BYTE *
2072    doxbufs(txt,bdtitle)
2073        BYTE *txt;
2074        BYTE *bdtitle;
2075    {
2076            BYTE cmds[CMDSLEN];
2077            BOOLEAN xiosvalid();
2078            EXTERN BYTE *clearit;
2079
2080            FOREVER {
2081                printf("%s",clearit);       /* clear screen */
2082                printf("%s\n",bdtitle);
2083                dbufs_display();                    /* display */
2084                printf("Drive (<CR> to exit) ? ");
2085                if( gets(cmds)!=cmds || *cmds==NULL ) {     /* they hit <CR>? */
2086                    if( xiosvalid() ) break;
2087                    USERR("Please correct drive buffers information\n");
2088                    press_return();
2089                    continue;
2090                }
2091                askabout(cmds);             /* go do what they want */
2092            }
2093            return NULLPTR;                 /* error msg for menu handler */
2094    }
2095
2096    /*************************************************************************/
2097    /* dbufs_display: display disk buffering info
2098    /*************************************************************************/
2099    dbufs_display()
2100    {
```

```
2101            REG WORD ii, jj, tot;           /* counters */
2102            DPH *dph;
2103            DINFO *dinf;
2104
2105            tot = 0;                        /* init this */
2106            dcalcsizes();                   /* check all the sizes */
2107            printf("\n\t*** Disk Buffering Information ***\n");
2108            printf("      Dir  Max/Proc   Data Max/Proc   Hash    Specified\n");
2109            printf("Drv   Bufs Dir Bufs   Bufs Dat Bufs   -ing    Buf Pgphs\n");
2110            printf("===   ==== ========   ==== ========   ====    =========\n");
2111            for( ii=0; ii<DPHNUM; ++ii ) {  /*** for each drive ***/
2112                if( dphtab[ii]==0 ) continue; /* no DP Hdr for this drive */
2113                dph = dh[ii]; dinf = di[ii];
2114                printf(" %c:   ",'A'+ii);   /* print which drive */
2115                dspdnums(dph->dph_dirbcb,dinf->d_ndirbs,dinf->d_pmdirbs);
2116                dspdnums(dph->dph_datbcb,dinf->d_ndatbs,dinf->d_pmdatbs);
2117                if( ! ASKFOR(dph->dph_hstbl) )
2118                    printf(" fixed  ");
2119                else if( dinf->d_hashing )
2120                    printf(" yes    ");
2121                else printf(" no     ");
2122                jj = dspdsize(dph,dinf);
2123                if( jj == -2 )
2124                    printf(" fixed   ");
2125                else if( jj == -1 )
2126                    printf(" ??      ");
2127                else {
2128                    printf(" %4.4x   ",jj >> 4);
2129                    tot += jj;
2130                }
2131                printf("\n");
2132            }                               /*** end for each drive ***/
2133            printf("Total paragraphs allocated to buffers: %x\n",tot >> 4);
2134    }
2135
2136
2137    /*************************************************************************/
2138    /* dspdsize: calculates a drive's user-requestable buffer size, in bytes */
2139    /*************************************************************************/
2140        WORD
2141    dspdsize(dph,dinf)                      /* calculate size requested */
2142        DPH *dph;                           /* returns -1 if still needs info */
2143        DINFO *dinf;                        /* returns -2 if pre-allocated */
2144    {
2145            REG WORD siz;
2146
2147            siz=0;
2148            if( !ASKFOR(dph->dph_dirbcb)  &&
2149                !ASKFOR(dph->dph_datbcb)  &&
2150                !ASKFOR(dph->dph_hstbl) )
2151                    return -2;              /* nothing to say here */
2152            if( ASKFOR(dph->dph_dirbcb) ){
2153                if( ASKFOR(dinf->d_ndirbs) )
2154                    return -1;
2155                if( !SHARED(dinf->d_ndirbs) )
2156                    siz += dinf->d_sdirbs * dinf->d_ndirbs;
2157            }
2158            if( ASKFOR(dph->dph_datbcb) ){
2159                if( ASKFOR(dinf->d_ndatbs) )
2160                    return -1;
```

```
2161                if( !SHARED(dinf->d_ndatbs) )
2162                    siz += dinf->d_sdatbs * dinf->d_ndatbs;
2163            }
2164            if( ASKFOR(dph->dph_hstbl) ){
2165                if( dinf->d_hashing )
2166                    siz += dinf->d_shash;
2167            }
2168            return siz;
2169    }
2170
2171
2172    /************************************************************************/
2173    /* dspdnums: displays the data/dir buf numbers                          */
2174    /************************************************************************/
2175    dspdnums(nfix,ninput,npm)               /* display subroutine */
2176        WORD nfix;
2177        WORD ninput;
2178        WORD npm;
2179    {
2180            if( !ASKFOR(nfix) )
2181                printf("fixed           ");
2182            else if( SHARED(ninput) )
2183                printf("shares %c:       ",'A'+GETSHARED(ninput));
2184            else {
2185                if( ASKFOR(ninput) )
2186                     printf(" ??  ");
2187                else printf(" %2.2x  ",ninput);
2188                if( ASKFOR(npm) )
2189                     printf("  ??       ");
2190                else printf("  %2.2x       ",npm);
2191            }
2192    }
2193
2194
2195    /************************************************************************/
2196    /* dcalcsizes: calculates buffer sizes across shared buffers            */
2197    /************************************************************************/
2198    dcalcsizes()                    /* make sure all the sizes are correct */
2199    {
2200            REG WORD ii, jj;
2201            REG WORD *pkk;
2202            DPH *dph;
2203            DINFO *dinf;
2204
2205            for( ii=0; ii<DPHNUM; ++ii )    /* init all sizes to zero */
2206                if(dphtab[ii]) {
2207                    dinf = di[ii];
2208                    dinf->d_sdirbs = dinf->d_sdatbs = 0;
2209                }
2210            for( ii=0; ii<DPHNUM; ++ii )            /* for each drive */
2211                if(dphtab[ii]) {
2212                    dph = dh[ii];
2213                    dinf = di[ii];
2214                    if( ASKFOR(dph->dph_dirbcb) )   /* max out the size of a dirbuf */
2215                        if( !ASKFOR(dinf->d_ndirbs) ) { /* if it's real data there */
2216                            if( SHARED(dinf->d_ndirbs) ) {
2217                                jj = GETSHARED(dinf->d_ndirbs);
2218                                pkk = & ((di[jj])->d_sdirbs);
2219                            } else
2220                                pkk = & dinf->d_sdirbs;
```

```
2221                        if( dinf->d_blksize > *pkk ) /* assign the maximum size found */
2222                            *pkk = dinf->d_blksize;
2223                    }
2224                if( ASKFOR(dph->dph_datbcb) )   /* max out the size of a data buf */
2225                    if( !ASKFOR(dinf->d_ndatbs) ) {
2226                        if( SHARED(dinf->d_ndatbs) ) {
2227                            jj = GETSHARED(dinf->d_ndatbs);
2228                            pkk = & ((di[jj])->d_sdatbs);
2229                        } else
2230                            pkk = & dinf->d_sdatbs;
2231                        if( dinf->d_blksize > *pkk )
2232                            *pkk = dinf->d_blksize;
2233                    }
2234            }
2235    }
2236
2237
2238    /************************************************************************/
2239    /* ask user about a drive                                               */
2240    /************************************************************************/
2241    askabout(drspec)                        /* ask user about a drive */
2242        BYTE *drspec;
2243    {
2244            REG WORD dr;
2245            DPH *dph;
2246            DINFO *dinf;
2247            BOOLEAN askhash();
2248
2249            if( *drspec == '*' ) {
2250                askstar();                  /* get info, apply to all disks */
2251                return;
2252            }
2253            dr = toupper(*drspec) - 'A';    /* which drive do they want to fix? */
2254            if( dr<0  ||  dr>=DPHNUM  ||  dphtab[dr]==0  ||  drspec[1] != ':') {
2255                USERR("Problem with '%s':\n",drspec);
2256                USERR("Please specify an existing drive between 'A:' and 'P:'\n");
2257                press_return();
2258                return;
2259            }
2260            dph = dh[dr];
2261            dinf = di[dr];
2262            if( dspdsize(dph,dinf) == -2 ) { /* is there anything to modify? */
2263                USERR("All buffers for %c: are fixed within the XIOS module.\n",
2264                    dr+'A');
2265                USERR("You can't modify this fixed information in GENSYS.\n");
2266                press_return();
2267                return;
2268            }
2269            if( ASKFOR(dph->dph_dirbcb) ){
2270                dinf->d_ndirbs = asknd(1,FALSE);    /* get directory info */
2271                dinf->d_pmdirbs = askpm(1,dinf->d_ndirbs);
2272            }
2273            if( ASKFOR(dph->dph_datbcb) ){
2274                dinf->d_ndatbs = asknd(2,(128==dinf->d_blksize)); /* get data info */
2275                dinf->d_pmdatbs = askpm(2,dinf->d_ndatbs);
2276            }
2277            if( ASKFOR(dph->dph_hstbl) ){
2278                dinf->d_hashing = askhash();
2279            }
2280    }
```

```
2281
2282
2283    /*************************************************************************/
2284    /* askstar: asks buffering questions & applies to all ASKFOR/unanswered drives
2285    /*************************************************************************/
2286    askstar()
2287    {
2288            REG WORD dr;
2289            DPH *dph;
2290            DINFO *dinf;
2291            WORD andirbs, andirpms, andatbs, andatpms;
2292            BOOLEAN ahash, askhash();
2293
2294            andirbs = asknd(1,FALSE);       /* Number of directory buffers? */
2295            andirpms = askpm(1,andirbs);    /* Process Max Dir Bufs? */
2296            andatbs = asknd(2,FALSE);       /* Number of data buffers? */
2297            andatpms = askpm(2,andatbs);    /* Process Max Data Bufs? */
2298            ahash = askhash();              /* Hashing? */
2299            for( dr=0; dr<DPHNUM; ++dr )
2300                if( dphtab[dr] ) {          /* for all existing drives */
2301                    dph = dh[dr]; dinf = di[dr];
2302                    if( ASKFOR(dph->dph_dirbcb) )
2303                        if( ASKFOR(dinf->d_ndirbs) ) {
2304                            dinf->d_ndirbs = andirbs;
2305                            dinf->d_pmdirbs = andirpms;
2306                        }
2307                    if( ASKFOR(dph->dph_datbcb) )
2308                        if( ASKFOR(dinf->d_ndatbs) ) {
2309                            dinf->d_ndatbs = andatbs;
2310                            dinf->d_pmdatbs = andatpms;
2311                        }
2312                    if( ASKFOR(dph->dph_hstbl) )
2313                        dinf->d_hashing = ahash;
2314                }
2315    }
2316
2317
2318
2319    /*************************************************************************/
2320    /* asknd: ask the user for the number of dir/data buffers
2321    /*************************************************************************/
2322        WORD
2323    asknd(type,zero_ok)
2324        WORD type;                          /* 1=dir, 2=data */
2325        BOOLEAN zero_ok;
2326    {
2327            REG WORD val, jj;
2328            BYTE *name;
2329            BYTE cmds[CMDSLEN];
2330            DPH *dph;
2331
2332            if( type==1 ) name="directory"; else name="data";
2333            FOREVER {
2334                printf("Number of %s buffers, or drive to share with ? ",name);
2335                gets( cmds );
2336                if( cmds[1] != ':' ) {
2337                    val = atoih(cmds);
2338                    if( val > 0  && val <= 127 ) break;
2339                        if(*cmds == '0') {
2340                            if( zero_ok ) break;
```

```
2341                        USERR("Number of buffers must be greater than 0\n");
2342                        press_return();
2343                        continue;
2344                    }
2345                } else {
2346                    val = toupper(*cmds) - 'A';     /* which drive do they want? */
2347                    if( val>=0 && val<DPHNUM && dphtab[val]!=0) { /* a drive maybe? */
2348                        dph = dh[val];
2349                        if( type==1 ) jj = dph->dph_dirbcb;
2350                        else          jj = dph->dph_datbcb;
2351                        if( ASKFOR(jj) ) {
2352                            if( type==1 ) jj = di[val]->d_ndirbs;
2353                            else          jj = di[val]->d_ndatbs;
2354                            if( !SHARED(jj) ) {
2355                                val |= SHRMSK;      /* turn on sharing */
2356                                break;
2357                            }
2358                        }
2359                        USERR("Drive %c: is not available for sharing\n",'A'+val);
2360                        press_return();
2361                        continue;
2362                    }
2363                }
2364                USERR("Please input a number, or an existing drive from 'A:' to 'P:'\n");
2365                press_return();
2366            }
2367            return val;
2368    }
2369
2370    /*************************************/
2371    /* askpm: ask about process maximum  */
2372    /*************************************/
2373        WORD
2374    askpm(type,pmmax)
2375        WORD type;
2376        WORD pmmax;
2377    {
2378            REG WORD val;
2379            BYTE *cmds[CMDSLEN];
2380            BYTE *name;
2381
2382            switch(type) { case 1: name="directory"; break; case 2: name="data"; }
2383            if( SHARED(pmmax) || pmmax<=0 )
2384                return 0;
2385            FOREVER {                       /* get input from user */
2386                printf("Maximum %s buffers per process [%x]? ",name,pmmax);
2387                gets( cmds );
2388                if( *cmds==NULL )
2389                     val = pmmax;           /* default value */
2390                else val = atoih(cmds);
2391                if( val > 0  &&  val <= pmmax ) break;
2392                USERR("Maximum must be > zero and <= %x\n",pmmax);
2393                press_return();
2394            }
2395            return val;
2396    }
2397
2398    /*****************************************/
2399    /* askhash: ask user if they'd like to hash */
2400    /*****************************************/
```

```
2401        BOOLEAN
2402    askhash()
2403    {
2404            REG WORD val;
2405            BYTE *cmds[CMDSLEN], *cp;
2406
2407            FOREVER {
2408                printf("Hashing [yes] ? ");
2409                gets( cmds );
2410                if( *cmds == NULL )
2411                    return TRUE;
2412                for( cp=cmds; *cp; ++cp ) {
2413                    if( *cp==' '  ||  *cp=='\t' ) {
2414                        *cp = NULL;
2415                        break;
2416                    }
2417                    if( isupper(*cp) )
2418                        *cp = tolower(*cp);
2419                }
2420                if( (val=whichof(cmds,",yes,true,on,hashing,no,false,off")) > 0 )
2421                    break;
2422                USERR("Please answer 'hashing', 'yes', or 'no'.\n");
2423                press_return();
2424            }
2425            return val<=4;                  /* TRUE iff answer was one of 1st four */
2426    }
2427
2428
2429
2430    /*************************************************************************/
2431    /* EXPORT                                                                */
2432    /* xiosvalid: validates that all the user-requestable info has been filled in
2433    /*************************************************************************/
2434        BOOLEAN
2435    xiosvalid()
2436    {
2437            REG WORD ii;
2438            REG WORD ndph;
2439
2440            ndph = 0;
2441            for( ii=0; ii<DPHNUM; ++ii )
2442                if( dphtab[ii]!=0 ) {
2443                    if( dspdsize(dh[ii],di[ii]) == -1 )
2444                        return FALSE;
2445                    ++ndph;
2446                }
2447            if( ndph==0 ) {
2448                USERR("no dph information in xios header...");
2449                exit(1);
2450            }
2451            return TRUE;
2452    }
2453
2454
2455
2456    /*************************************************************************/
2457    /* EXPORT                                                                */
2458    /* genxios: generates disk buffers (to be placed after the rest of the
2459    /*      system tables.
2460    /*************************************************************************/
```

```
2461    #define OFF_BCBPTR 0xA
2462    #define OFF_DPHNTAB 0x12
2463    ADDR locrsvd_off;                       /* local reserved offset, for DIRJJF fix */
2464
2465        ADDR                                /* returns final offset */
2466    genxios(doffset)                        /* generate the disk buffers */
2467        ADDR doffset;                       /* offset in data file */
2468    {
2469            REG WORD ii, jj;                /* counter */
2470            REG BOOLEAN dm;                 /* modified flag */
2471            REG ADDR lastlink;              /* used for threading lists */
2472            ADDR locbufs_off;               /* local buffer offset for DIRBUF grouping */
2473            LONG dmagic;                    /* where word needing fix goes into file */
2474            BCB bcb;                        /* place for organizing, calc for fixups */
2475            BCBH bcbh;
2476            DPH *dph;
2477            DINFO *dinf;
2478            BOOLEAN xiosvalid();
2479
2480            if( !xiosvalid() ) CRASH("invalid disk buffer information");
2481
2482            for( ii=0; ii<DPHNUM; ++ii )    /* allocate CHECK SUM VECTORS */
2483                if( dh[ii] ) {              /* if there's something to look at */
2484                    dph = dh[ii];           /* convenient ptr */
2485                    dinf = di[ii];
2486                    if( ASKFOR(dph->dph_csv) ){
2487                        dinf->d_modified = TRUE;
2488                        if( dinf->d_csvsize == 0 )
2489                            dph->dph_csv=0;
2490                        else {
2491                            dph->dph_csv = doffset;
2492                            doffset += bufwrite(dinf->d_csvsize,ZBYTE);
2493                        }
2494                    }
2495                }
2496            for( ii=0; ii<DPHNUM; ++ii )    /* allocate ALLOCATION TABLE VECTORS */
2497                if( dh[ii] ) {              /* if there's something to look at */
2498                    dph = dh[ii];           /* convenient ptr */
2499                    dinf = di[ii];
2500                    if( ASKFOR(dph->dph_alv) ){
2501                        dinf->d_modified = TRUE;
2502                        dph->dph_alv = doffset;
2503                        doffset += bufwrite(dinf->d_alvsize,ZBYTE);
2504                    }
2505                }
2506            locbufs_off = 0;                /* amount (bytes) of local dir buffer*/
2507            for( ii=0; ii<DPHNUM; ++ii )    /* allocate DIR BCBs & BUFFERS */
2508                if( dh[ii] ) {              /* if there's something to look at */
2509                    dph = dh[ii];           /* convenient ptr */
2510                    dinf = di[ii];
2511                    if( ASKFOR(dph->dph_dirbcb) ){  /* DIR BCBs */
2512                        dinf->d_modified = TRUE;
2513                        if( ! SHARED(dinf->d_ndirbs) ) { /* take care of sharing later */
2514                            lastlink=0;                 /* linker */
2515                            zfill(&bcb,sizeof(BCB));    /* init to zeroes */
2516                            bcb.bcb_drv = 0xFF;
2517                            for( jj=dinf->d_ndirbs; jj>0; --jj ){
2518                                bcb.bcb_link = lastlink;
2519                                /* pts to loc of buffer relative to locrsvd_off */
2520                                bcb.bcb_bufptr = locbufs_off;
```

```
2521                            locbufs_off += dinf->d_sdirbs; /* increase buf size */
2522                            /* calc where bcb will be written to, for fixlater */
2523                            dmagic = lseek(fns,0L,1) + OFF_BCBPTR;
2524                            fixlater(dmagic,F_ADD,&locrsvd_off);
2525                            /* finally, write out the bcb info */
2526                            lastlink = doffset;       /* point to the bcb */
2527                            doffset += datwrite(&bcb,sizeof bcb);
2528                        }
2529                        bcbh.bcbh_lr = lastlink; /* fill in the bcb hdr */
2530                        bcbh.bcbh_pm = dinf->d_pmdirbs;
2531                        dph->dph_dirbcb = doffset; /* link to head of list */
2532                        /* write out a buffer control block hdr */
2533                        doffset += datwrite(&bcbh,sizeof bcbh);
2534                    }
2535                }
2536            }
2537        for( ii=0; ii<DPHNUM; ++ii )    /* allocate DATA BCBs & BUFFERS */
2538            if( dh[ii] ) {              /* if there's something to look at */
2539                dph = dh[ii];           /* convenient ptr */
2540                dinf = di[ii];
2541                /* the bufs can go out of the system page, but the bcbs need to be in */
2542                if( ASKFOR(dph->dph_datbcb) ){  /* DATA BCBs: !!temporary!! */
2543                    dinf->d_modified = TRUE;
2544                    if( ! SHARED(dinf->d_ndatbs) ) { /* do sharing later */
2545                        lastlink=0;                  /* linker */
2546                        zfill(&bcb,sizeof(BCB));     /* init to zeroes */
2547                        bcb.bcb_drv = 0xFF;
2548                        for( jj=dinf->d_ndatbs; jj>0; --jj ){
2549                            bcb.bcb_link = lastlink;
2550                            bcb.bcb_bufptr = bufs_seg; /* point into reserved area */
2551                            bufs_seg += (dinf->d_sdatbs) >>4; /* cvt to pgphs */
2552                            /* calculate where the bcb.bcb_bufptr will be written in file */
2553                            dmagic = lseek(fns,0L,1) + OFF_BCBPTR;
2554                            fixlater(dmagic,F_ADD,&rsvd_seg); /* fix the bufptr */
2555                            lastlink = doffset;
2556                            /* write out a buffer control block */
2557                            doffset += datwrite(&bcb,sizeof bcb);
2558                        }
2559                        bcbh.bcbh_lr = lastlink; /* fill in the bcb hdr */
2560                        bcbh.bcbh_pm = dinf->d_pmdatbs;
2561                        dph->dph_datbcb = doffset; /* link to head of list */
2562                        /* write out a buffer control block hdr */
2563                        doffset += datwrite(&bcbh,sizeof bcbh);
2564                    }
2565                }
2566            }
2567        /* now write the local directory buffers */
2568        locrsvd_off = doffset;  /* address of directory buffer area */
2569        doffset += bufwrite(locbufs_off,ZBYTE); /* write out the buffers */
2570
2571        /* dmagic: magic number to add to dphtab[ii] */
2572        dmagic = OFF_DPHHTAB + fldstart;/* where in file does SYSDAT start */
2573
2574        for( ii=0; ii<DPHNUM; ++ii )    /* allocate HASH TABLES */
2575            if( dh[ii] ) {              /* if there's something to look at */
2576                dph = dh[ii];           /* convenient ptr */
2577                dinf = di[ii];
2578                if( ASKFOR(dph->dph_hstbl) ) {  /* HASH TABLE ALLOC */
2579                    dinf->d_modified = TRUE;
2580                    if( dinf->d_hashing ) {
```

```
2581                        dph->dph_hstbl = bufs_seg; /* point into reserved area */
2582                        bufs_seg += (dinf->d_shash) >>4; /* cvt bytes to pgphs */
2583                        /* remember where the dph_hstbl word will be stored */
2584                        /* so we can adjust bufs_seg by starting loc of rsvd_seg */
2585                        fixlater(dmagic+dphtab[ii],F_ADD,&rsvd_seg);
2586                    } else dph->dph_hstbl = 0;
2587                }
2588            }                           /**** end of processing ****/
2589
2590        for( ii=0; ii<DPHNUM; ++ii )    /* resolve shared buffers */
2591            if( dh[ii] ) {
2592                if( ASKFOR(dh[ii]->dph_dirbcb) ) {
2593                    jj = GETSHARED(di[ii]->d_ndirbs);
2594                    dh[ii]->dph_dirbcb = dh[jj]->dph_dirbcb;
2595                }
2596                if( ASKFOR(dh[ii]->dph_datbcb) ) {
2597                    jj = GETSHARED(di[ii]->d_ndatbs);
2598                    dh[ii]->dph_datbcb = dh[jj]->dph_datbcb;
2599                }
2600            }                           /* end resolve shared buffers */
2601
2602        return doffset;
2603    }
2604
2605
2606    /*************************************************************************/
2607    /* EXPORT                                                                */
2608    /* fixupxios_info: does the fixup for the dph info in the xios           */
2609    /*************************************************************************/
2610    fixupxios_info()                /* call from fixups */
2611    {
2612        REG WORD ii;                /* counter */
2613        REG LONG dmagic;            /* addr var */
2614
2615        for( ii=0; ii<DPHNUM; ++ii )
2616            if( di[ii]->d_modified ) {
2617                dmagic = fldstart + dphtab[ii];
2618                lseek(fns,dmagic,0); /* where to write to */
2619                write(fns,dh[ii],sizeof(DPH));  /* what to write */
2620            }
2621    }
2622
2623
2624
2625
2626    /*************************************************************************/
2627    /*************************************************************************/
2628    /*      M E N U  handling routines                                       */
2629    /*                                                                       */
2630    /*      Written by Bill Fitler                                           */
2631    /*************************************************************************/
2632    /*************************************************************************/
2633    #ifndef MAINMODULE
2634    #include <portab.h>
2635    #include <stdio.h>
2636    #include <menu.h>
2637    EXTERN BOOLEAN verbose;         /* flag; be wordy */
2638    #endif
2639
2640    MLOCAL BOOLEAN eflag = FALSE;   /* user error */
```

```
2641
2642    /*************************************************************************/
2643    /* bldmenu: builds a menu list from successive calls                     */
2644    /*************************************************************************/
2645        MENU *
2646    bldmenu(muroot,typ,ptr,name,description)
2647        MENU *muroot;                   /* root of menu list (or NULLPTR) */
2648        WORD typ;                       /* type of menu item */
2649        BYTE *ptr;                      /* pointer to something... */
2650        BYTE *name;                     /* command name for this menu item */
2651        BYTE *description;              /* long label for this item */
2652    {
2653            REG MENU *mu;               /* ptr to created item */
2654            REG MENU *tmu;              /* temp ptr for linked list scan */
2655
2656            mu = (MENU *)malloc(sizeof(*mu));       /* allocate memory for item */
2657            mu->mutype = typ;
2658            mu->muiptr = ptr;
2659            mu->muname = name;
2660            mu->mudesc = description;
2661            mu->munext = NULLPTR;
2662            if( muroot == NULLPTR )         /* if NULL list */
2663                    return mu;              /* return something to start with */
2664            for( tmu=muroot; tmu->munext != NULLPTR; tmu = tmu->munext )
2665                    ;                       /* append to end of list */
2666            tmu->munext = mu;
2667            return muroot;                  /* keep list in order */
2668    }
2669
2670    /*************************************************************************/
2671    /* domenu: executes the specified commands (in cbuf) from the menu (mnu) */
2672    /*************************************************************************/
2673    #define CMDLEN 80
2674    #define USERR ++eflag;printf
2675    #define ECHK(fn) _errchk(fn,cmd,cmdval)
2676
2677        VOID
2678    domenu(cbuf,mnu)
2679        BYTE *cbuf;                     /* command buffer */
2680        MENU *mnu;                      /* table of menu specs */
2681    {
2682            REG MENU *mi;               /* ptr to menu item */
2683            LOCAL BYTE cmd[CMDLEN];     /* place for command & value */
2684            LOCAL BYTE *cmdval;         /* ptr within cmd */
2685            LOCAL BYTE *cp;             /* temp char ptr */
2686            BYTE *msetbool();           /* set a boolean value */
2687            BYTE *msetbyte();           /* set a numeric byte value */
2688            BYTE *msetword();           /* set an integer pointer */
2689            BYTE *msettxt();            /* set a text value */
2690            BYTE *(*proc)();            /* procedure pointer */
2691            BYTE *msetdrv();            /* set a drive value */
2692            BYTE *_nxtcmd();            /* command parser */
2693            MENU *_fndcmd();            /* command lookup */
2694
2695            eflag = FALSE;              /* no errors yet */
2696            while((cbuf=_nxtcmd(cbuf,cmd))){ /* while there are commands in cbuf */
2697                cmdval = NULLPTR;       /* init value ptr for command */
2698                for( cp=cmd; *cp; cp++ ) {  /* scan command for assign op */
2699                    if( isupper(*cp) )      /* convert all commands to LOWER CASE */
2700                        *cp = tolower(*cp);
```

```
2701                    if( *cp == '=' ) {          /* found? */
2702                        *cp++ = NULL;           /* terminate command */
2703                        cmdval = cp;            /* point value to after assign op */
2704                        break;                  /* and terminate loop */
2705                    }
2706                }
2707                if( (mi=_fndcmd(cmd,mnu)) == NULLPTR ){ /* scan menu commands */
2708                    USERR("'%s' is not a command for this menu\n",cmd);
2709                } else {                        /******** cmd loop ********/
2710                    switch(mi->mutype) {        /* handle cmd type */
2711                    case MBOOL:                 /** BOOLEAN command **/
2712                        ECHK(msetbool(mi->muiptr,cmdval));
2713                        break;
2714                    case MBYTE:                 /** BYTE command **/
2715                        ECHK(msetbyte(mi->muiptr,cmdval));
2716                        break;
2717                    case MWORD:                 /** INTEGER command **/
2718                        ECHK(msetword(mi->muiptr,cmdval));
2719                        break;
2720                    case MTEXT:                 /** TEXT command **/
2721                        ECHK(msettxt(mi->muiptr,cmdval));
2722                        break;
2723                    case MPROC:                 /** PROCEDURAL command **/
2724                        proc = mi->muiptr;      /* assign to function ptr */
2725                        ECHK(((*proc)(cmdval,mi->mudesc)));
2726                        break;
2727                    case MDRIV:                 /** DRIVE letter command **/
2728                        ECHK(msetdrv(mi->muiptr,cmdval));
2729                        break;
2730                    }                           /* end switch */
2731                }                               /********* end cmd loop *********/
2732                if(eflag) {                     /* user error? */
2733                    press_return();
2734                    *cbuf = NULL;
2735                } else if(verbose)              /* are we being wordy? */
2736                    prtmival(mi);               /* let the user know what happened */
2737            }                                   /**** end while loop ****/
2738    }
2739
2740
2741
2742    /* _nxtcmd: parses cin into cout */
2743        BYTE *
2744    _nxtcmd(cin,cout)                           /* returns ptr to after next cmd */
2745        BYTE *cin;                              /* command in buf */
2746        BYTE *cout;                             /* command out buf */
2747    {
2748            /***** handle quoted strings??? *****/
2749            while( isspace(*cin) )              /* skip leading spaces */
2750                    ++cin;
2751            if( *cin == NULL )                  /* check for EOS */
2752                    return NULLPTR;
2753            while( *cin && !isspace(*cin) ) /* scan to eos or whitespace */
2754                    *cout++ = *cin++;           /* xfer to command out buf */
2755            *cout = NULL;                       /* null terminate command out buf */
2756            return cin;                         /* return ptr to after command */
2757    }
2758
2759
2760
```

```
2761    /* _fndcmd: scans thru menu structure "mu" looking for command name "cn" */
2762    /*              returns ptr to menu item or NULLPTR */
2763        MENU *
2764    _fndcmd(cn,mus)
2765        BYTE *cn;                       /* command name */
2766        MENU *mus;                      /* ptr to menu list */
2767    {
2768            REG WORD cnl;               /* cmd name len */
2769            MENU *musave;               /* extra ptr to menu item */
2770
2771            cnl = strlen(cn);
2772            for( ; ; mus = mus->munext) {   /* scan menu list */
2773                if( mus==NULLPTR )          /* if we hit the end then fail */
2774                    return NULLPTR;
2775                if( strncmp(cn,mus->muname,cnl) == 0 )      /* cn prefix of name?*/
2776                    break;                  /* yes, we have a candidate */
2777            }
2778            musave = mus;
2779            for( mus=mus->munext; ; mus=mus->munext ) { /* finish scanning list */
2780                if( mus==NJLLPTR )          /* if we hit the end then succeed */
2781                    return musave;          /* we reached end of list: found it! */
2782                if( strncmp(cn,mus->muname,cnl) == 0 )
2783                    return NULLPTR;         /* another match: ambiguous command */
2784            }
2785    }
2786
2787
2788
2789    /* _errchk: checks for error returns on functions */
2790        VOID
2791    _errchk(emsg,cm,cmv)
2792        BYTE *emsg;                     /* error message to check */
2793        BYTE *cm;                       /* command generating the error */
2794        BYTE *cmv;                      /* value generating the error */
2795    {
2796            if( emsg ) {                /* non NULL error message return? */
2797                    USERR("Error on command `%s",cm);
2798                    if(cmv)
2799                            USERR("=%s",cmv);
2800                    USERR("': %s\n",emsg);
2801            }
2802    }
2803
2804
2805
2806    /* msetbool: sets the BOOLEAN pointed to by "bp" according to value in "cvp" */
2807        BYTE *                          /* return NULLPTR, or ptr to err msg */
2808    msetbool(bp,cvp)
2809        BYTE *bp;                       /* pointer to boolean to set */
2810        BYTE *cvp;                      /* pointer to value to set to */
2811    {
2812            REG WORD bv;
2813            REG BYTE *tp;
2814
2815            if( cvp==NULLPTR )          /* no val: toggle boolean */
2816                    bv = (*bp ? 4 : 1);     /* toggle by setting bv index */
2817            else {
2818                for( tp=cvp; *tp; ++tp )
2819                    if( isupper(*tp) ) *tp=tolower(*tp);
2820                if( (bv=whichof(cvp,",on,yes,true,off,no,false"))==0 )
```

```
2821                    return "value must be 'yes' or 'no'";
2822            }
2823            if( bv < 4 )
2824                    *bp = -1;               /* zap to 0xFFFF */
2825            else    *bp = 0;                /* zap to zero */
2826            return NULLPTR;
2827    }
2828
2829
2830        BYTE *
2831    msetbyte(bp,cvp)                        /* set numeric BYTE value */
2832        BYTE *bp;                           /* ptr to BYTE to set */
2833        BYTE *cvp;                          /* value to set BYTE to */
2834    {
2835            REG WORD val;                   /* temp holding place */
2836
2837            val=0;
2838            if(cvp != NULLPTR) val=atoih(cvp);      /* assume numbers are in HEX!! */
2839            if( cvp==NULLPTR || (val==0  &&  !isdigit(*cvp)) )
2840                return "value must be a number";
2841            if( val > 255 )
2842                return "value must be less than FF hex (255 decimal)";
2843            *bp = val;
2844            return NULLPTR;
2845    }
2846
2847
2848        BYTE *
2849    msetword(ip,cvp)
2850        WORD *ip;                           /* ptr to WORD to set */
2851        BYTE *cvp;                          /* value to set WORD to */
2852    {
2853            REG WORD val;                   /* temp holding place */
2854
2855            val=0;
2856            if(cvp != NULLPTR) val=atoih(cvp);      /* assume numbers are in HEX!! */
2857            if( cvp==NULLPTR || (val==0  &&  !isdigit(*cvp)) )
2858                return "value must be an unsigned hex number between 0 and FFFFh";
2859            *ip = val;
2860            return NULLPTR;
2861    }
2862
2863
2864        WORD
2865    atoih(cp)                               /* conver ascii hex to word */
2866        BYTE *cp;                           /* buffer number */
2867    {
2868            REG WORD v;                     /* resulting value */
2869            REG BYTE cv;                    /* character value */
2870            BYTE *savcp;                    /* save orig ptr */
2871
2872            v = 0; savcp = cp;              /* init value */
2873            for( cv = *cp; ; cv = *++cp ){  /* for each char in cp */
2874                if( isdigit(cv) )
2875                    v = (v<<4) + (cv-'0');  /* convert digit and add */
2876                else {                      /* maybe it's a letter */
2877                    if( isupper(cv) )       /* upper case letter? */
2878                        cv = tolower(cv);   /* convert it */
2879                    if( cv<'a' || cv>'f')   /* range check */
2880                        break;              /* terminate loop if failed */
```

```
2881                    v = (v<<4)+(10+cv-'a'); /* convert letter and add */
2882                }
2883            }
2884            if( cp-savcp <= 4 )
2885                return v;                   /* return the value */
2886            else {
2887                *savcp |= 0x80;
2888                return 0;           /* turn on hi bit, make isdigit test fail */
2889            }
2890    }
2891
2892        BYTE *
2893    msettxt(tp,cvp)
2894        BYTE **tp;                          /* place to store ptr to value */
2895        BYTE *cvp;                          /* value to set BYTEs to */
2896    {
2897            REG BYTE *tmp;                  /* temp ptr for allocated storage */
2898
2899            if( cvp==NULLPTR )
2900                cvp="";                     /* point to null string, instead */
2901            tmp = malloc(1+strlen(cvp));    /* get space for string */
2902            strcpy(tmp,cvp);                /* copy to allocated space */
2903            *tp = tmp;                      /* assign ptr */
2904            return NULLPTR;
2905    }
2906
2907        BYTE *
2908    msetdrv(dp,cvp)
2909        BYTE *dp;                           /* place to store ptr to value */
2910        BYTE *cvp;                          /* value to set DRIVE to */
2911    {
2912            REG WORD v;                     /* work place */
2913
2914            if( cvp!=NULLPTR ) {            /* check for drive letter */
2915                if( isupper(*cvp) )
2916                    *cvp = tolower(*cvp);
2917                v = *cvp - 'a';             /* convert drive spec to nibble */
2918            }
2919            if( cvp==NULLPTR  ||  v<0  ||  v>15  || cvp[1] != ':' )
2920                return "you must specify a drive 'A:' thru 'P:'";
2921            *dp = v;                        /* looks okay, set it */
2922            return NULLPTR;                 /* no errors */
2923    }
2924
2925
2926    /*************************************************************************/
2927    /* prtmenu: displays the menu labels, values and descriptors             */
2928    /*************************************************************************/
2929        VOID
2930    prtmenu(mtitle,mnu)
2931        BYTE *mtitle;                       /* title of menu */
2932        MENU *mnu;                          /* table specifying menu */
2933    {
2934            printf("\n\n%s\n",mtitle);      /* give the menu a title */
2935            for( ; mnu!=NULLPTR; mnu = mnu->munext ) /* travel down list */
2936                prtmival(mnu);
2937    }
2938
2939
2940    prtmival(mnu)                           /* return ptr to value of item */
```

```
2941        MENU *mnu;                          /* item to get value of */
2942    {
2943            REG BYTE *ival;
2944            BYTE ivbuf[10];                 /* local buf for display */
2945            BYTE *vbp;                      /* value byte ptr */
2946            WORD *vwp;                      /* value word ptr */
2947            WORD v;                         /* place to put value */
2948
2949            printf("%12.12s ",mnu->muname); /* first the name */
2950            ival = ivbuf; *ival = NULL;     /* init value */
2951            switch(mnu->mutype) {   /* now the value */
2952                case MBOOL:
2953                    if( *(mnu->muiptr) )
2954                        ival="[Y]";
2955                    else ival="[N]";
2956                    break;
2957                case MBYTE:
2958                    vbp = mnu->muiptr;
2959                    v = (*vbp & 0xFF);
2960                    sprintf(ivbuf,"[%2.2x]",v);     /* display in hex */
2961                    break;
2962                case MWORD:
2963                    vwp = mnu->muiptr;
2964                    v = *vwp;
2965                    sprintf(ivbuf,"[%4.4x]",v);     /* display in hex */
2966                    break;
2967                case MTEXT:
2968                    vbp = *((BYTE **)mnu->muiptr);  /* go get the char ptr */
2969                    if( strlen(vbp) > 4 )
2970                        sprintf(ivbuf,"[%-4.4s>",vbp);
2971                    else sprintf(ivbuf,"[%s]",vbp);
2972                    break;
2973                case MPROC:
2974                    /* leave blank */
2975                    break;
2976                case MDRIV:
2977                    sprintf(ivbuf,"[%c:]",'A'+(*mnu->muiptr));
2978                    break;
2979            }
2980            printf("%-6s ",ival);
2981            printf("%s\n",mnu->mudesc);     /* finally the description */
2982    }
2983
2984
2985    /*************************************************************************
2986    /* whichof: scans the string 'set' (delimited by 1st char in 'set')
2987    /*      for 1st occurrence of string 'sample'.
2988    /*      Returns 0 if 'sample' not a prefix of any item in set or if
2989    /*              'sample' is a prefix of more than one item in set;
2990    /*              delimiter# o.w.
2991    /* E.G. if set = ",on,yes,true,off,no,false"  then
2992    /*      whichof("yes",set) == 2;
2993    /*      whichof("y",set) == 2;
2994    /*      whichof("o",set) == 0;
2995    /*      whichof("x",set) == 0;
2996    /* NOTE: results not guaranteed if item delimiter (*set) is in string 'sample',
2997    /*      or if null item is in set (adjacent delimiters).
2998    /*************************************************************************/
2999
3000    /* first, a subroutine: */
```

```
3001    /* _wget: returns NULLPTR if sa not prefix of item in st; */
3002    /*        returns end of item (ptr in st) otherwise */
3003    BYTE *_wget(sa,st,de)
3004    BYTE *sa;
3005    BYTE *st;
3006    BYTE de;
3007    {
3008            BYTE *s2;
3009
3010            for( ;; ) {                                     /* do forever */
3011                    for( s2=sa; *s2 && *s2 == *st; ){
3012                            ++s2; ++st;                     /* try match */
3013                    }
3014                    if( *s2 == NULL ){      /* success! we've matched all of sa */
3015                            while( *st && *st != de ) ++st; /* adv to nxt delim */
3016                            return( --st );                 /* backup one char */
3017                    }
3018                    while( *st  &&  *st != de ) /* no match, look for next item */
3019                            ++st;                           /* adv to nxt delim */
3020                    if( *st == NULL )
3021                            return( NULLPTR );              /* end of list */
3022                    ++st;
3023            }
3024    }
3025
3026    WORD whichof(sample,set)
3027    BYTE *sample;
3028    BYTE *set;
3029    {
3030            WORD sx;
3031            BYTE *sp, *np;
3032
3033            if( (sp=_wget(sample,set+1,*set)) == NULLPTR )
3034                    return 0;                               /* not found */
3035            if( _wget(sample,sp+1,*set) != NULLPTR )
3036                    return 0;                               /* ambiguous */
3037            for( sx=0, np=set; np<sp; np++ )
3038                    if( *np == *set ) sx++;         /* count delimiters */
3039            return sx;
3040    }
3041
3042
3043
3044
3045    /*************************************************************************/
3046    /*************************************************************************/
3047    /* misc.c:                                                               */
3048    /* some miscellaneous functions which belong in the system library */
3049    /*************************************************************************/
3050    /*************************************************************************/
3051    #ifndef MAINMODULE
3052    #include <portab.h>
3053    #include <stdio.h>
3054    #include <cpm.h>                        /* directory search support */
3055    #endif
3056
3057    #define FCB struct fcbstruct
3058    FCB {
3059            BYTE    fcbdr;                  /* drive */
3060            BYTE    fcbname[8];             /* name */
```

```
3061            BYTE    fcbext[3];              /* extension */
3062            BYTE    fcbrest[24];            /* all the rest */
3063    };
3064
3065
3066    #define FSEARCh(faddr) (bdos(17,faddr)&0xFF)
3067    #define NSEARCH(faddr) (bdos(18)&0xFF)  /* doesn't use FCB */
3068    #define SETDMA(dma) bdos(26,dma)        /* where to put dir info */
3069    #define SECSIZE 128
3070    #define AMASK 0x7F
3071
3072    /* dirsearch: returns a DLIST of names matching 'fs' */
3073        DLIST *                             /* returns list of names found in */
3074    dirsearch(fs)                           /* directory search for */
3075        BYTE *fs;                           /* file spec */
3076    {
3077            FCB f, *fp;                     /* use this FCB */
3078            REG WORD ret;                   /* ptr */
3079            DLIST *_newdli(), *ls;          /* list */
3080            BYTE dmabuf[SECSIZE];           /* place for directory info */
3081
3082            fp = &f;                        /* everything else uses address */
3083            parsefn(fs,fp);                 /* fill fcb with filespec */
3084            SETDMA(dmabuf);                 /* tell em where to put it */
3085            if( (ret=FSEARCH(fp)) > 3 )     /* look for first occurrence */
3086                return NULLPTR;             /* no luck */
3087            ls=_newdli(dmabuf+(ret<<5),NULLPTR);    /* start off a list */
3088            while( (ret=NSEARCH(fp)) <= 3 ) { /* do all the search_nexts */
3089                ls = _newdli(dmabuf+(ret<<5),ls); /* link them together */
3090            }
3091            return ls;
3092    }
3093
3094
3095        DLIST *
3096    _newdli(f,lnk)
3097        FCB *f;
3098        DLIST *lnk;
3099    {
3100            REG WORD i;
3101            BYTE *np, nb[15];               /* name buffer */
3102            DLIST *newl;                    /* new list item */
3103
3104                                            /* make a pretty filename */
3105            np = nb;
3106            for( i=0; i<11; ++i )           /* zap out any high bits in filename */
3107                f->fcbname[i] &= AMASK;     /* mask off attribute bits */
3108            for( i=0; i<8; ++i ) {          /* handle name spec */
3109                if( f->fcbname[i] == ' ' )  /* end of name? */
3110                    break;                  /* okay */
3111                *np++ = f->fcbname[i];      /* o.w. stuff into name buffer */
3112            }
3113            *np++ = '.';                    /* tack this on */
3114            for( i=0; i<3; ++i ){           /* handle ext spec */
3115                if( f->fcbext[i] == ' ' )   /* end of ext? */
3116                    break;
3117                *np++ = f->fcbext[i];
3118            }
3119            *np++ = NULL;                   /* null terminate string */
3120            /*****************************************************/
```

```
3121
3122            np = malloc(np-nb);             /* save it away */
3123            strcpy(np,nb);                  /* do the copy */
3124            newl = (DLIST *)malloc(sizeof(*newl));  /* get ptr area set up */
3125            newl->dlname = np;
3126            newl->dlnext =  lnk;
3127            return newl;
3128    }
3129
3130
3131
3132    parsefn(fs,f)
3133        BYTE *fs;                           /* file spec */
3134        FCB *f;                             /* fcb */
3135    {
3136            REG WORD i;
3137            REG BYTE *cp;
3138
3139            for( cp=fs; *cp; cp++ )         /* make everything upper case */
3140                if( islower(*cp) )
3141                    *cp = toupper(*cp);
3142            for( i=0; i<24; ++i )           /* init fcb */
3143                f->fcbrest[i] = 0;
3144            if( *(fs+1) == ':' ) {          /* drive spec? */
3145                f->fcbdr = 1 + *fs - 'A';   /* 0=default, 1 = A:,... */
3146                fs += 2;                    /* skip dr spec */
3147            } else f->fcbdr = 0;            /* default drive */
3148            for( i=0; *fs && *fs != '.' && i<8; fs++, ++i ) {       /* NAME. */
3149                if( *fs == '*' ) {          /* wildcard? */
3150                    for( ; i<8; ++i )       /* fill up rest of name */
3151                        f->fcbname[i]='?';  /* with search spec */
3152                    for( ; *fs && *fs!='.'; fs++ )  /* ignore to end or '.' */
3153                        ;
3154                    break;                  /* terminate loop */
3155                }
3156                f->fcbname[i] = *fs;
3157            }
3158            for( ; i<8; ++i )               /* fill up rest of name */
3159                f->fcbname[i] = ' ';        /* with blanks */
3160            for( ; *fs; fs++ )              /* skip rest of name */
3161                if( *fs == '.' ) {          /* end of name? */
3162                    fs++; break;            /* yes, so exit this loop */
3163                }
3164            for( i=0; *fs && i<3; fs++,++i ) {                      /* .EXT */
3165                if( *fs == '*' ){           /* wildcard */
3166                    for( ; i<3; ++i )
3167                        f->fcbext[i] = '?';
3168                    break;
3169                }
3170                f->fcbext[i] = *fs;
3171            }
3172            for( ; i<3; ++i )               /* fill up rest of ext */
3173                f->fcbext[i] = ' ';         /* with blanks */
3174    }
3175
3176
3177
3178    /* access: tests for existence: return NULL if exist, -1 o.w. */
3179        WORD
3180    access(fn)
```

```
3181        BYTE *fn;
3182    {
3183            FCB f;                          /* use this FCB */
3184            BYTE dmabuf[SECSIZE];           /* place for directory info */
3185
3186            parsefn(fn,&f);                 /* fill fcb with filespec */
3187            SETDMA(dmabuf);                 /* tell em where to put it */
3188            if( FSEARCH(&f) > 3 )           /* look for first occurrence */
3189                return -1;                  /* no luck */
3190            return 0;
3191    }
3192
3193        BYTE *
3194    gets(s)
3195        BYTE *s;
3196    {
3197            BYTE *sav;
3198
3199            if( fgets(s,32767,stdin) == NULLPTR )
3200                    return NULLPTR;
3201            if( !isatty(0) )                /* reading from CON:? */
3202                    fputs(s,stdout);        /* echo if not */
3203            for( sav=s; *s && *s != '\n'; ++s )
3204                    ;
3205            *s = '\0';
3206            return sav;
3207    }
3208
3209    /*************************************************************************/
3210        BYTE
3211    getdrive()
3212    {
3213            return bdos(25,0);
3214    }
3215
3216
3217    /*************************************************************************/
3218
3219    /* press_return: let user look at screen */
3220    press_return()
3221    {
3222            BYTE bitbucket[40];             /* leave room */
3223
3224            if( !isatty(0) ) {              /* stdin from CON:? */
3225                printf("Error in command file: terminating program\n");
3226                exit(1);
3227            }
3228            printf("\07Press RETURN to continue ");
3229            if( gets(bitbucket)!=bitbucket )
3230                    exit(1);                /* eof, quit now */
3231    }
3232
3233    /* patch: provides a patch area */
3234    patch()                                 /* should never get called */
3235    {
3236            int i;
3237            i=i+5; i=i+5; i=i+5; i=i+5; i=i+5;
3238            i=i+5; i=i+5; i=i+5; i=i+5; i=i+5;
3239            i=i+5; i=i+5; i=i+5; i=i+5; i=i+5;
3240            i=i+5; i=i+5; i=i+5; i=i+5; i=i+5;
```

```
3241            i=i+5; i=i+5; i=i+5; i=i+5; i=i+5;
3242            i=i+5; i=i+5; i=i+5; i=i+5; i=i+5;
3243            i=i+5; i=i+5; i=i+5; i=i+5; i=i+5;
3244            i=i+5; i=i+5; i=i+5; i=i+5; i=i+5;
3245            i=i+5; i=i+5; i=i+5; i=i+5; i=i+5;
3246            i=i+5; i=i+5; i=i+5; i=i+5; i=i+5;
3247            return;
3248    }
3249
3250
3251    /*************************************************************************/
3252    /*************************************************************************/
3253    /*                          T H E   E N D                                */
3254    /*************************************************************************/
3255    /*************************************************************************/
```



```
**** CROSS REFERENCE *****
          ADDR 1905,1910,1912,1913,1914,1915,1916,1937,1950,1980,2004,2463,2465,
               2467,2471,2472
         ALIGN  318,1313
         AMASK 3070,3107
         AREAD   81
        ASKFOR 2010,2117,2148,2149,2150,2152,2153,2158,2159,2164,2180,2185,2188,
               2214,2215,2224,2225,2269,2273,2277,2302,2303,2307,2308,2312,2351,
               2486,2500,2511,2542,2578,2592,2596
        AWRITE   83
           BCB 1941,1942,2474,2515,2546
          BCBH 1935,1936,2475
       BDOSMOD  190, 402
       BIGWORD 1353,1359
        BROPEN  128, 144, 152, 408, 876,1625
        BUFSIZ  892, 909
       BWCREAT  368
   BYTE_ADJUST  317,1314,1411,1415
           CCB 1398
          CCPM   30,  34, 143, 240, 283, 324, 342, 410, 479, 517, 629,1471,1490,
               1535
        CIOMOD  189, 401
        CMDLEN 2673,2683
       CMDSLEN   56, 646, 674, 702, 983,2076,2329,2379,2405
         CRASH  133, 847, 859, 877, 896, 904, 906, 910,1409,1865,2480
         DINFO 1955,1956,2006,2028,2042,2043,2103,2143,2203,2246,2290,2477
         DLIST  121, 727, 743, 745, 781, 817,1394,1767,1782,1816,3073,3079,3095,
               3098,3102,3124
           DPB 1920,1921,2026,2053
           DPH 1908,1909,2005,2027,2041,2045,2102,2142,2202,2245,2289,2330,2476,
               2619
        DPHNUM 2003,2005,2006,2036,2048,2111,2205,2210,2254,2299,2347,2441,2482,
               2496,2507,2537,2574,2590,2615
          ECHK 2675,2712,2715,2718,2721,2725,2728
           FCB 3057,3058,3077,3097,3134,3183
        FCHECK  392, 498, 553,1404,1526
           FIX 1834,1841,1843,1853
          FLAG 1400
       FSEARCH 3066,3085,3188
         F_ADD 1866,2524,2554,2585
         F_PUT  458, 459, 460,1540
    GETSHARED 2013,2183,2217,2227,2593,2597
         GROUP  494, 844, 893
```

```
          buf1  550, 560, 561, 562
          buf2  550, 561, 562
      bufs_seg 1407,1537,1541,1543,2550,2551,2581,2582
      bufwrite 1989,2492,2503,2569
            bv 2812,2816,2820,2823
           byt 1989,1991,1996
          cbuf   99, 102, 104, 105, 106,2678,2679,2696,2734
            cc 1815,1818,1820,1821,1822,1824,1825,1826,2013
     cc_mimic 1466
    cc_msource 1466
           ccb 1398,1465,1466,1468
            cg  494, 524, 525, 526, 527, 528, 529, 533
      chkaddr 1354,1421,1493
    chkmodules  123, 128, 180
           cin 2744,2745,2749,2750,2751,2753,2754,2756
       clbdos  402, 422, 442
        clclo  401, 422, 441
      clearit   61,  62,  76,  84, 648, 653, 676, 681, 704, 709, 986, 995,1769,
                1772,2078,2081
       clearx   25,  61,  84
        clmem  400, 422, 440
        clnet  404, 406, 422
    clos_label  396, 397, 422, 438
        close   80,  82, 142, 149, 158, 432, 559, 879,1644
        clrtm  399, 422, 439
        clsup  395, 398, 422, 438
      cltotal  422, 426, 474, 526, 527, 533
       clxios  416, 420, 422, 444
            cm 2791,2793,2797
           cmd 2675,2683,2696,2698,2707,2708
       cmdhdr  393, 473, 523, 524, 529, 535
          cmds   56,  66,  68,  79,  87,  89, 646, 656, 657, 659, 674, 684, 685,
                 687, 702, 713, 714, 716, 983, 999,1008,2076,2085,2091,2329,2335,
                2336,2337,2339,2346,2379,2387,2388,2390,2405,2409,2410,2412,2420
       cmdval  977, 978, 989, 990, 993,2675,2684,2697,2703,2712,2715,2718,2721,
                2725,2728
           cmv 2791,2794,2798,2799
            cn 2764,2765,2771,2775,2782
           cnl 2768,2771,2775,2782
     cntmlist  338, 981, 988, 991,1009,1132,1408
     copyargs   66,  99
     copynote   22
    copyright 1822
         count 1384,1429,1438,1447,1456,1467,1485,1508
          cout 2744,2746,2754,2755
            cp  726, 732, 733, 734, 762, 768, 769, 770,2405,2412,2413,2414,2417,
                2418,2685,2698,2699,2700,2701,2702,2703,2865,2866,2872,2873,2884,
                3137,3139,3140,3141
           cpm 3054
           cpp  960, 962, 963, 965, 966
         cseek 1979,2032,2044,2052
          csfd 1976,1982,2030
        csgoff 1977,1982,2031
            cv 2869,2873,2874,2875,2877,2878,2879,2881
           cvp 2808,2810,2815,2818,2820,2831,2833,2838,2839,2849,2851,2856,2857,
                2893,2895,2899,2900,2901,2902,2908,2910,2914,2915,2916,2917,2919
    d_alvsize 1966,2056,2503
    d_blksize 1967,2054,2221,2222,2231,2232,2274
    d_csvsize 1968,2055,2488,2492
```

```
      dogensys  131, 154, 287, 372
          doit   71, 309
         dolbl  222, 288,1763
         domem  223, 269, 977
        domenu   68,  89, 651, 659, 679, 687, 707, 716, 990,1008,2678
    donew_xios  476
         dorsp  221, 290, 698
        dosysp  220, 252, 670
       doxbufs  224, 284,2072
        doxios  219, 275, 642
            dp 2908,2909,2921
       dpb_blm 1924
       dpb_bsh 1923
       dpb_cks 1929,2055
       dpb_dav 1928
       dpb_drm 1927,2057
       dpb_dsm 1926,2056
       dpb_exm 1925
      dpb_ltem 1920
       dpb_off 1930
       dpb_phm 1932
       dpb_psh 1931,2054
       dpb_spt 1922
           dph 2027,2050,2052,2102,2113,2115,2116,2117,2122,2141,2142,2148,2149,
               2150,2152,2158,2164,2202,2212,2214,2224,2245,2260,2262,2269,2273,
               2277,2289,2301,2302,2307,2312,2330,2348,2349,2350,2476,2484,2486,
               2489,2491,2498,2500,2502,2509,2511,2531,2539,2542,2561,2576,2578,
               2581,2586
       dph_alv 1914,2500,2502
       dph_csv 1913,2486,2489,2491
    dph_datbcb 1916,2116,2149,2158,2224,2273,2307,2350,2542,2561,2596,2598
    dph_dirbcb 1915,2115,2148,2152,2214,2269,2302,2349,2511,2531,2592,2594
       dph_dpb 1912,2052
      dph_hstbl 1917,2117,2150,2164,2277,2312,2578,2581,2586
       dph_ltem 1908
      dph_space 1911
       dph_xlt 1910
        dphtab 2004,2034,2037,2044,2112,2206,2211,2254,2300,2347,2442,2585,2617
           dps  843,  848,  853,  857,  858
            dr 2244,2253,2254,2260,2261,2264,2288,2299,2300,2301
        drspec 2241,2242,2249,2253,2254,2255
        drvmsg  215,.233,  234,  248
     dsotables  504,  505,1420
      dspdnums 2115,2116,2175
      dspdsize 2122,2141,2262,2443
      dspmlist  512,  996,1106
       dsstart  426,  458,  503,  504,  505,1496,1528,1532
         edsys  374,  470
      edsysver   35,   38,   59,  169,  555
         eflag 2640,2674,2695,2732
          emsg 2791,2792,2796,2800
            ep  726,  731,  732,  762,  767,  768
       ep_eoff  449,  459
       ep_eseg  437,  438,  439,  440,  441,  442,  444,  460
       ep_ioff  447,  448,  449
       ep_iseg  437,  438,  439,  440,  441,  442,  444
          exit  166,  172,  370,  413,  557,1346,1363,1627,2449,3226,3230
             f 1022,1035,1038,1043,1045,1050,1054,3077,3082,3096,3097,3107,3109,
               3111,3115,3117,3132,3134,3143,3145,3147,3151,3156,3159,3167,3170,
```

```
              290, 295, 298
        main   52
     mainmenu   42,  68,  77,  89, 298
   mainmtitle   41,  77, 231
  make_sysfile   92, 362
       malloc  233, 235,1049,1219,1245,1782,1783,1843,2041,2042,2656,2901,3122,
              3124
           mb 1768,1780,1783,1784
           md 1395,1428,1430,1432,1558,1561,1565,1567,1570,1574,1575,1576
    md_length 1565,1570
      md_link 1430,1574
     md_start 1575
           me  211, 237, 270, 271, 300
      memmenu   44, 300, 985, 990, 997,1008
      memroot  338, 512, 984, 988, 991, 996,1009,1024,1053,1067,1073,1077,1085,
              1208,1214,1408,1562
     memtitle  977, 979, 997
         menu 2636
           mi 1181,1183,1189,1193,1194,2682,2707,2710,2712,2715,2718,2721,2724,
              2725,2728,2736
      mlfirst 1050,1116,1118,1120,1122,1144,1150,1151,1158,1160,1167,1189,1215,
              1220,1226,1236,1243,1246,1254,1563,1564
       mllast 1051,1116,1118,1120,1122,1145,1150,1151,1152,1158,1160,1167,1216,
              1218,1221,1227,1238,1247,1253,1563,1570
       mlnext.1074,1089,1090,1097,1119,1142,1146,1153,1190,1191,1193,1223,1224,
              1230,1239,1249,1251,1257,1263,1562,1571
       mlsize 1052,1116,1118,1122,1143,1160,1167,1220,1222,1226,1228,1243,1244,
              1248,1253,1255,1563,1565
           mm 1066,1085,1088,1089,1185,1188,1190,1194,1209,1214,1230,1239,1250,
              1251,1257,1263
           mn 1066,1085,1088,1092,1094,1095,1109,1115,1116,1118,1119,1120,1122,
              1135,1142,1143,1144,1145,1146,1150,1151,1152,1153,1158,1160,1164,
              1166,1167,1186,1188,1189,1190,1191,1193,1209,1214,1215,1216,1218,
              1220,1221,1222,1223,1224,1226,1227,1228,1230,1231,1236,1238,1239,
              1240,1243,1244,1249,1253,1254,1255,1257,1258,1263,1386,1555,1562,
              1563,1564,1565,1570,1571
         mnew 1023,1049,1050,1051,1052,1053
          mnu 2678,2680,2707,2930,2932,2935,2936,2940,2941,2949,2951,2953,2958,
              2963,2968,2977,2981
           mo 1066,1085,1086,1089,1090,1095,1096,1109,1115,1119,1120,1135,1142,
              1143,1144,1145,1146,1147,1150,1151,1152,1153,1154,1158,1164,1209,
              1219,1220,1221,1222,1223,1224,1245,1246,1247,1248,1249,1250,1251
       module  869, 870, 876
           mp 1066,1073,1074,1075,1094,1096,1097,1098
           mq 1066,1073,1074,1096,1097
        mroot 1106,1107,1115,1132,1133,1142,1181,1182,1188,1195
           ms 1017,1018,1029,1035,1062,1063,1072,1080,1081,1083
     msetbool 2686,2712,2808
     msetbyte 2687,2715,2831
      msetdrv 2691,2728,2908
      msettxt 2689,2721,2893
     msetword 2688,2718,2849
       mtitle 2930,2931,2934
           mu 2653,2656,2657,2658,2659,2660,2661,2663,2666
       mudesc 2660,2725,2981
       muiptr 2658,2712,2715,2718,2721,2724,2728,2953,2958,2963,2968,2977
       muname 2659,2775,2782,2949
       munext 2661,2664,2666,2772,2779,2935
       muroot 2646,2647,2662,2664,2667
```

```
          ptr 2646,2649,2658
    pure_xios  415, 419, 429, 446
      putname 1683,1705,1728,1738
      qbuflen  139, 266,1503,1504,1514
      qc_link 1448
          qcb 1397,1446,1448,1450
           qs  781, 791, 792, 794
            r  213, 237, 291, 292, 302
         read  134, 156, 850, 899, 910,1657,1867,2033,2045,2053
       rename  561, 562
          ret 3078,3085,3087,3088,3089
           rh 1610,1622,1623,1624,1657,1686,1709,1731
         rhdr 1605,1622,1623,1624,1659,1661,1663,1665,1681,1684,1703,1707,1726,
              1729
           rp  816, 817, 822, 827
          rpd 1606,1623,1624,1631,1637,1658,1683,1705,1706,1728
           rr 1289,1291,1293
           rs  723, 724, 729, 731, 732, 745, 750, 751, 759, 760, 765, 767, 768,
               781, 784, 785, 791, 792, 793, 794, 795,1394,1529,1530
      rs_link 1599,1684,1707,1729
   rs_ncopies 1601,1663,1665,1681,1703,1726
   rs_sdatvar 1600,1659,1661
     rs_space 1602
  rsp_display  710, 816
   rsp_header 1597
     rsp_last 1310,1527,1531,1684,1685,1707,1708,1729,1730
      rsp_seg 1311,1528,1532,1534,1679,1680,1685,1687,1688,1700,1702,1708,1710,
              1712,1725,1730,1732
   rspexclude  228, 292, 759
   rspinclude  227, 291, 723
      rspjoin  737, 742
      rspmenu   46, 302, 703, 707, 711, 716
      rspname 1612,1656,1660,1664,1682,1683,1704,1705,1727,1728
      rspnbuf 1611,1658,1660,1664
        rspnm 1616,1617,1625,1626,1641
         rsps  121, 140, 710, 747, 748, 750, 784, 788, 791,1529
       rspzap  763, 771, 778
     rsvd_seg  506,1534,1537,1543,2554,2585
         ruda 1607,1624,1679,1680,1700,1702,1724,1725
           rv  874, 878, 880,1653,1661,1665,1667
            s  212, 237, 254, 255, 256, 257, 258, 259, 260, 262, 263, 264, 265,
               266, 301,1022,1036,1043,1045,1047,1052,1054,3194,3195,3199,3202,
              3203,3205
           s1  802, 803, 805, 806, 808
           s2  802, 803, 805, 806, 808,3008,3011,3012,3014
           sa 3003,3004,3011
    sa_length 1503
     sa_start 1499,1502
        saddr 1388,1557,1564,1570,1575,1577
      sample 3026,3027,3033,3035
         sat 1401,1497,1498,1499,1500,1502,1503,1505,1507,1509
         sav 3197,3203,3206
       savcp 2870,2872,2884,2887
  sd_bdosmod  441, 442, 588
      sd_ccb  606,1464,1474
   sd_ciomod  440, 441, 587
    sd_cmode  257, 634
  sd_dayfile  256, 602
   sd_endseg  507, 510, 593,1543
```

```
        stdout 3202
           str 1801,1802,1806,1807
        strcat  105, 106, 185, 186, 187, 188, 189, 190, 192, 194
        strcpy 1784,2902,3123
        strlen  555,1806,2771,2901,2969
       strncmp 2775,2782
       strncpy 1658
            sx 3030,3037,3038,3039
        sysdat  581, 582, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594,
                595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607,
                608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620,
                621, 622, 623, 624, 626, 627, 630, 631, 632, 634
       syslbls 1771,1774,1787,1823
      syspmenu   45, 301, 675, 679, 682, 687
      sysvalid   57,  72, 321
           tmp 2897,2901,2902,2903
           tmu 2654,2664,2666
            to 1738,1739,1753
       tolower 2418,2700,2819,2878,2916
           tot 2101,2105,2129,2133
        totmds 1412,1415,1429
       totnump 1138,1140,1163,1170,1174
       totopen  138, 263, 330, 334,1456
        totpds 1410,1411,1412,1438
       totqcbs  137, 265,1416,1447
       toupper  806,2253,2346,3141
            tp 2813,2818,2819,2893,2894,2903
        trimit 1226,1244,1253,1269
      trimlist  510,1204
            tx  957, 958
           txt 1763,1764,2072,2073
           typ 2646,2648,2657
          type 2323,2324,2332,2349,2352,2374,2375,2382
         uaddr 1354,1355,1359
        ucount 1385,1514,1515
     ud_csinit 1680,1700,1724
     ud_dsinit 1679,1702,1724
     ud_esinit 1679,1702,1724
     ud_ssinit 1679,1702,1725
        unlink  367, 389, 560
             v 2868,2872,2875,2881,2885,2912,2917,2919,2921,2947,2959,2960,2964,
               2965
           val 2327,2337,2338,2346,2347,2348,2352,2353,2355,2359,2367,2378,2389,
               2390,2391,2395,2404,2420,2425,2835,2837,2838,2839,2841,2843,2853,
               2855,2856,2857,2859
           vbp 2945,2958,2959,2968,2969,2970,2971
       verbose   67, 242, 246, 391, 472, 497, 500, 552,1403,1525,2637,2735
       version 1821
           vwp 2946,2963,2964
       whichof 2420,2820,3026
        wrapup   95, 548
         write  393, 473, 535, 539, 555, 911,1326,1343,1576,1686,1709,1731,1807,
               1808,1872,1996,2619
      writelbl  396,1813
             x  210, 237, 277, 278, 279, 280, 281, 299
            xf 2021,2022,2030,2033,2045,2053
xfer_mixed_rsp 1638,1717
 xfer_pure_rsp 1633,1694
      xfer_rsp 1530,1616
```

```
xfer_separate_rsp 1632,1672
       xferbuf  892, 910, 911
       xfergrp  395, 399, 400, 401, 402, 404, 416, 427, 474, 475, 869
  xferpart_grp  430, 431, 878, 885,1687,1688,1710,1712,1732
      xiosmenu   43, 299, 647, 651, 654, 659
     xiosvalid  325, 343,2077,2086,2435,2478,2480
         xlist  123, 128, 160, 163
       xmtitle  642, 644, 654, 698, 700, 711
            xt 1472,1473,1474,1475,1476,1477,1479,1536,1537,1540,2033,2034
      xt_alloc 1536,1537,1540
        xt_ccb 1474
     xt_dphtab 2034
        xt_lcb 1477
      xt_nccbs 1473
      xt_nlcbs 1475,1476
      xt_nvcns 1472
  xt_ticks_sec 1479
       xtrapds  136, 264,1410
            xx  317, 318, 891, 905, 906, 907, 908, 913,1313
            yy  891, 897, 902, 905, 908, 909,1314
       zero_ok 2323,2325,2340
         zfill 1428,1437,1446,1455,1465,1498,1507,1512,1561,1586,2043,2515,2546
            zz 1210,1226,1228,1244,1248,1253,1255
**** end cross reference ***
```

```
/*****************************************************************************/
/*****************************************************************************/
/* edsys.h  -  the include module(s) for genccpm.c
/*****************************************************************************/
/*****************************************************************************/

/***** include files *****/
#include <portab.h>
#include <stdio.h>
/*************************/

/*****************************************/
/* compiler differences                  */
/* configured for Computer Innovations C86 */
#define BROPEN(fn) open(fn,BREAD)       /* binary open for read */
#define BWCREAT(fn) creat(fn,BUPDATE)   /* binary creat for write */
LONG lseek();
BYTE *malloc();
BYTE *gets();
#define NULLPTRI 0                      /* NULLPTR initializer */

/* compile time options: assume the "-dXXX" flag is used, where XXX is: */

/* #define CCPM                         /* this is a CCP/M EDSYS */
/* #define MPM                          /* this is an MP/M EDSYS */
/* #define NET                          /* there's a NET module to load! */

/*****************************************************************************/
#ifndef CCPM
#ifndef MPM
        ERROR: you must specify one of "MPM" or "CCPM"
#endif
#endif
#ifdef CCPM
#ifdef MPM
        ERROR: these compile options are mutually exclusive: only 1 allowed!
#endif
#endif



/*****************************************************************************/
/*****************************************************************************/
/* menu.h : include with progs using menu rtns
/*****************************************************************************/
/*****************************************************************************/

#define MENU struct menuitem
MENU {
        WORD mutype;            /* type of menu item */
        BYTE *muiptr;           /* ptr to information for menu item */
        BYTE *muname;           /* name of menu item */
        BYTE *mudesc;           /* description of menu item */
        MENU *munext;           /* ptr to next menu item */
};

#define MBOOL 1
#define MBYTE 2
#define MWORD 3
#define MTEXT 4
```

```
  61    #define MPROC 5
  62    #define MDRIV 6
  63
  64
  65
  66    /*************************************************************************/
  67    /*************************************************************************/
  68    /* cpm.h - support for cpm types of calls
  69    /*************************************************************************/
  70    /*************************************************************************/
  71    #define SECSIZ 128
  72    #define BUFSIZ 512      /* should be in stdio.h */
  73
  74    /*************************************************************************/
  75    /* directory search support */
  76    #define DLIST struct dliststr           /* directory list structure */
  77    DLIST {
  78            BYTE *dlname;                   /* file name */
  79            DLIST *dlnext;                  /* next item in list */
  80    };
  81
  82    /*************************************************************************/
  83    /* .CMD header record Group structure */
  84    #define GROUP struct cmdgrp
  85    GROUP {                                 /* what a Cmd Hdr group looks like */
  86            BYTE gtype;                     /* type of group (see below) */
  87            UWORD glen;                     /* length in paragraphs of group */
  88            JWORD gabase;                   /* pgph addr for non-relocatable grp */
  89            UWORD gmin;                     /* min pgphs to alloc to group */
  90            UWORD gmax;                     /* max pgphs to alloc to group */
  91    };
  92    #define GTYPCODE 1                      /* Code Group */
  93    #define GTYPDATA 2                      /* Data Group */
  94
  95
  96
  97    /*************************************************************************/
  98    /*************************************************************************/
  99    /* mpm.h - data structures peculiar to MP/M & Concurrent
 100    /*************************************************************************/
 101    /*************************************************************************/
 102    #define MINKMEM 80              /* minimum memory allocation in paragraphs */
 103
 104    #define XIOSTAB struct xios_table
 105    XIOSTAB {
 106            BYTE xt_tick;
 107            BYTE xt_ticks_sec;
 108            BYTE xt_door;
 109            BYTE xt_rsvd[2];
 110            BYTE xt_nvcns;
 111            BYTE xt_nccbs;
 112            BYTE xt_nlcbs;
 113            UWORD xt_ccb;
 114            UWORD xt_lcb;
 115            UWORD xt_dphtab[16];
 116            UWORD xt_alloc;
 117    };
 118    #define XTABLEN (sizeof(XIOSTAB))
 119    #define XTABLOC   (0X0c0cL)
 120    #define XTABALLOC (0X0c38L)
```

```
121
122     #define MD struct mem_descriptor
123     MD {
124             UWORD md_link;
125             UWORD md_start;
126             UWORD md_length;
127             UWORD md_plist;
128             UWORD md_unused;
129     };
130
131     #define PD struct proc_descriptor
132     PD {
133             UWORD pd_link;
134             UWORD pd_thread;
135             BYTE pd_stat;
136             BYTE pd_prior;
137             UWORD pd_flag;
138             BYTE pd_name[8];
139             UWORD pd_uda;
140             BYTE pd_dsk;
141             BYTE pd_user;
142             BYTE pd_ldsk;
143             BYTE pd_luser;
144             UWORD pd_mem;
145             BYTE pd_restofit[24];
146     };
147
148     #define LOCK struct lock_item
149     LOCK {
150             UWORD lo_link;
151             UWORD lo_restofit[4];
152     };
153
154     #define UDA struct uda_item
155     UDA {
156             BYTE ud_1space[0x50];
157             UWORD ud_csinit;
158             UWORD ud_dsinit;
159             UWORD ud_esinit;
160             UWORD ud_ssinit;
161             BYTE ud_2space[0xA8];           /* uda is 0x100 bytes long */
162     };
163
164     #define QCB struct qcb_descriptor
165     QCB {
166             UWORD qc_link;
167             BYTE qc_restofit[26];
168     };
169
170     #define FLAG struct flag_item
171     FLAG {
172             BYTE fl_status;
173             UWORD fl_pd;
174     };
175
176     #define CCB struct ccb_item
177     CCB {
178             UWORD cc_attach;
179             UWORD cc_queue;
180             BYTE cc_flag;
```

```
181             BYTE cc_startcol;
182             BYTE cc_column;
183             BYTE cc_nchar;
184             BYTE cc_mimic;
185             BYTE cc_msource;
186             BYTE cc_type;
187             BYTE cc_xdev;
188     };
189
190     #define SATITEM struct sat_item
191     SATITEM {
192             UWORD sa_start;
193             UWORD sa_length;
194             BYTE sa_nall;
195     };
196
197
198
199
200     /*****************************************************************************/
201     /*****************************************************************************/
202     /* here's the body of edsys.h
203     /*****************************************************************************/
204     /*****************************************************************************/
205     #ifdef CCPM
206     #define PROGNAME "GENCCPM"
207     #define VERLABEL "Concurrent CP/M-86 2.0"
208     #define VERSION 0x1420
209     #define SYSFILE "CCPM.SYS"
210     #define NEWSYS  "CCPM.$Y$"
211     #define OLDSYS  "CCPM.OLD"
212     #define MODEXT  ".CON"
213     #define SUPMOD  "SUP.CON"
214     #define RTMMOD  "RTM.CON"
215     #define MEMMOD  "MEM.CON"
216     #define CIOMOD  "CIO.CON"
217     #define BDOSMOD "BDOS.CON"
218     #define NETMOD  "NET86.CON"
219     #define SYSMOD  "SYSDAT.CON"
220     #define XIOSMOD "XIOS.CON"
221     #endif
222
223     #ifdef MPM
224     #define PROGNAME "GENMPM"
225     #define VERLABEL "MP/M-86 2.1"
226     #define VERSION 0x1121
227     #define SYSFILE "MPM.SYS"
228     #define NEWSYS  "MPM.$Y$"
229     #define OLDSYS  "MPM.OLD"
230     #define MODEXT  ".MPM"
231     #define SUPMOD  "SUP.MPM"
232     #define RTMMOD  "RTM.MPM"
233     #define MEMMOD  "MEM.MPM"
234     #define CIOMOD  "CIO.MPM"
235     #define BDOSMOD "BDOS.MPM"
236     #define NETMOD  "NET86.MPM"
237     #define SYSMOD  "SYSDAT.MPM"
238     #define XIOSMOD "XIOS.MPM"
239     #endif
240
```

```
241
242     /*****************************************************************************/
243     #define USERR printf                    /* how to handle user errors */
244     #define CRASH(fn) {fprintf(stderr,fn);exit(1);}
245     #define FCHECK(sp) /* disable */
246             /* if(ferror(sp)){printf("Error on output: out of space?");exit(1);} */
247     #define BDOSPRT printf
248     #define CMDSLEN 80
249     #define PROMPT "Changes? "
250
251
252     /* memory list definition */
253     #define MLIST struct memorylist
254     MLIST {
255             UWORD mlfirst;                  /* addr of 1st pgph */
256             UWORD mllast;                   /* addr of last pgph */
257             WORD mlsize;                    /* partition size in K bytes */
258             MLIST *mlnext;                  /* ptr to next item in list */
259     };
260
261     /* ENTRY point structure decls */
262     #define ENTRY struct entrypoint
263     ENTRY {
264             UWORD ep_eoff;                  /* entry offset */
265             UWORD ep_eseg;                  /* entry segment */
266             UWORD ep_ioff;                  /* init offset */
267             UWORD ep_iseg;                  /* init segment */
268     };
269
270     /* FIXup structure decls */
271     #define FIX struct fixitem
272     FIX {
273             LONG f_addr;                    /* offset in file */
274             WORD f_type;                    /* either F_ADD or F_PUT */
275             WORD *f_ptr;                    /* ptr to value to add or put */
276             FIX *f_next;                    /* next fixup item */
277     };
278     #define F_ADD 1                         /* add value of value to word in file*/
279     #define F_PUT 2                         /* put value ... */
280
281
282     /*****************************************************************************/
283     /* Program Globals:
284     /*      Keep all the globals defined in one file, for ease of access
285     /*      Definitions first, then declarations
286     /*****************************************************************************/
287
288     /******** definitions for the MAINMODULE **********/
289     #ifdef MAINMODULE
290
291     /*****************************************************************************/
292     /* Edsys Program Globals
293     /*****************************************************************************/
294         GLOBAL BYTE *copyright = "Copyright (C) 1983, Digital Research";
295         GLOBAL BYTE *version = VERLABEL;
296         GLOBAL BYTE *clearit;               /* Home cursor & erase screen */
297         GLOBAL BOOLEAN verbose=TRUE;        /* flag to tell EDSYS to be wordy */
298         GLOBAL BOOLEAN doit=FALSE;          /* flag: tell EDSYS to do the rest of it */
299         GLOBAL WORD fns = -1;               /* File descriptor for New System file */
300         GLOBAL BOOLEAN dogensys = FALSE;    /* Gen System from scratch? */
```

```
301         GLOBAL BOOLEAN doclean = FALSE;     /* Clean SYSFILE first from destination drive? */
302         GLOBAL BOOLEAN donew_xios = FALSE;  /* edit in a NEW XIOS? */
303         GLOBAL BOOLEAN pure_xios = FALSE;   /* XIOS groups are pure (separate) code & data? */
304         GLOBAL BYTE destdrv;                /* where the SYSFILE goes */
305
306     /*************************************************************************/
307     /* Sysfile Globals
308     /*************************************************************************/
309         GLOBAL WORD totmds;                 /* total number of memory descriptors */
310         GLOBAL WORD totpds;                 /* total number of process descriptors */
311         GLOBAL WORD xtrapds;                /* number of pds wanted by user */
312         GLOBAL WORD totqcbs;                /* tot num q control blocks */
313         GLOBAL UWORD qbuflen;               /* length in bytes of q buf area */
314         GLOBAL WORD totopen;                /* total open files and locked records */
315
316         GLOBAL DLIST *rsps=NULLPTR1;        /* list of RSP file names */
317         GLOBAL DLIST *syslbls=NULLPTR1;     /* lines of OS label */
318         GLOBAL MLIST *memroot=NULLPTR1;     /* memory partitions */
319         GLOBAL XIOSTAB xt;                  /* xios info table */
320
321     #define SDLEN 0x210
322         GLOBAL BYTE sdat_area[SDLEN];       /* where to store the SYSDAT info */
323         GLOBAL BYTE cmdhdr[SECSIZ];         /* command header for NEWSYS */
324
325     /* module length information: lengths are in paragraphs */
326         GLOBAL UWORD clsup;                 /* code length: supervisor */
327         GLOBAL UWORD clos_label;            /* code length: OS label */
328         GLOBAL UWORD clrtm;                 /*  "     "   : real time monitor */
329         GLOBAL UWORD clmem;                 /*  "     "   : memory manager */
330         GLOBAL UWORD clcio;                 /*  "     "   : char i/o */
331         GLOBAL UWORD clbdos;                /*  "     "   : bdos */
332         GLOBAL UWORD clnet;                 /*  "     "   : network handler */
333         GLOBAL UWORD clxios;                /*  "     "   : xios */
334         GLOBAL UWORD cltotal;               /*  "     "    of all modules */
335         GLOBAL UWORD dsstart;               /* starting segment of system data */
336         GLOBAL UWORD dlsysdat;              /* data length: system data */
337         GLOBAL UWORD dlxios;                /* length of xios in data seg */
338         GLOBAL UWORD dltables;              /* length of tables */
339         GLOBAL UWORD dsotables;             /* starting offset of tables */
340         GLOBAL UWORD dlrsps;                /* length of rsps */
341         GLOBAL UWORD dltotal;               /* length of data, xios, tables & rsps */
342         GLOBAL UWORD rsvd_seg;              /* start of OS reserved (buffer) area (not in load image) */
343         GLOBAL UWORD bufs_seg;              /* ptr to allocated buffer at rsvd_seg */
344
345     /*** fixup locations ***/
346         GLOBAL LONG fldstart;               /* location of SYSDAT start */
347         GLOBAL LONG locmfl;                 /* location in file of mem free list */
348
349     /*************************************************************************/
350     /* system data variables for MP/M-86 or Concurrent CP/M-86 */
351     /*************************************************************************/
352         GLOBAL ENTRY * sd_supmod;
353         GLOBAL ENTRY * sd_rtmmod;
354         GLOBAL ENTRY * sd_memmod;
355         GLOBAL ENTRY * sd_ciomod;
356         GLOBAL ENTRY * sd_bdosmod;
357         GLOBAL ENTRY * sd_xiosmod;
358         GLOBAL ENTRY * sd_netmod;
359         GLOBAL UWORD * sd_mpmseg;
360         GLOBAL UWORD * sd_rspseg;
```

```
        GLOBAL UWORD * sd_endseg;
        GLOBAL BYTE * sd_module_map;
        GLOBAL BYTE * sd_ncns;
        GLOBAL BYTE * sd_nlst;
        GLOBAL BYTE * sd_nccb;
        GLOBAL UBYTE * sd_nflags;
        GLOBAL BYTE * sd_srchdisk;
        GLOBAL UWORD * sd_mmp;
        GLOBAL BYTE * sd_nslaves;
        GLOBAL BYTE * sd_dayfile;
        GLOBAL BYTE * sd_tempdisk;
        GLOBAL BYTE * sd_tickspsec;
        GLOBAL UWORD * sd_lul;
        GLOBAL UWORD * sd_ccb;
        GLOBAL UWORD * sd_flags;
        GLOBAL UWORD * sd_mdul;
        GLOBAL BYTE * sd_nxmds;
        GLOBAL UWORD * sd_mfl;
        GLOBAL BYTE * sd_nmparts;
        GLOBAL UWORD * sd_pul;
        GLOBAL BYTE * sd_nxpd;
        GLOBAL UWORD * sd_qul;
        GLOBAL BYTE * sd_nqds;
        GLOBAL UWORD * sd_qmau;
        GLOBAL UWORD * sd_qmastart;          /* segment start of q buffer */
        GLOBAL UWORD * sd_qmalen;            /* length of q buffer */
        GLOBAL UWORD * sd_verptr;            /* pts to SUP segment ver string */
        GLOBAL UWORD * sd_vn;                /* F.12 version num */
        GLOBAL UWORD * sd_mpmvn;             /* F.163 version num */
        GLOBAL BYTE * sd_ncondev;
        GLOBAL BYTE * sd_nlstdev;
        GLOBAL BYTE * sd_nciodev;
        GLOBAL UWORD * sd_lcb;
        GLOBAL BYTE * sd_plock_max;
        GLOBAL BYTE * sd_popen_max;
        GLOBAL BYTE * sd_cmode;

    #else

    /****** declarations for non MAINMODULE *******/

    /*****************************************************************************/
    /* Program Globals
    /*****************************************************************************/
        EXTERN BYTE *copyright;
        EXTERN BYTE *version;
        EXTERN BYTE *clearit;                /* Home cursor & erase screen */
        EXTERN BOOLEAN verbose;              /* flag to tell EDSYS to be wordy */
        EXTERN BOOLEAN doit;                 /* flag: tell EDSYS to do the rest of it */
        EXTERN WORD fns;                     /* File descriptor for New System file */
        EXTERN BOOLEAN dogensys;             /* Gen System from scratch? */
        EXTERN BOOLEAN doclean;              /* Clean SYSFILE first from destination drive? */
        EXTERN BOOLEAN donew_xios;           /* edit in a NEW XIOS? */
        EXTERN BOOLEAN pure_xios;            /* XIOS groups are pure (separate) code & data? */
        EXTERN BYTE destdrv;                 /* where the SYSFILE goes */

    /*****************************************************************************/
    /* Sysfile Globals
    /*****************************************************************************/
```

```
421         EXTERN WORD totmds;             /* total number of memory descriptors */
422         EXTERN WORD totpds;             /* total number of process descriptors */
423         EXTERN WORD xtrapds;            /* number of pds wanted by user */
424         EXTERN WORD totqcbs;            /* tot num q control blocks */
425         EXTERN UWORD qbuflen;           /* length in bytes of q buf area */
426         EXTERN WORD totopen;            /* total open files & locked records */
427
428         EXTERN DLIST *rsps;             /* list of RSP file names */
429         EXTERN DLIST *syslbls;          /* lines of OS label */
430         EXTERN MLIST *memroot;          /* memory partitions */
431         EXTERN XIOSTAB xt;              /* xios info table */
432
433         EXTERN BYTE sdat_area[];        /* where to store the SYSDAT info */
434         EXTERN BYTE *cmdhdr;            /* command header for NEWSYS */
435
436         EXTERN UWORD clsup;             /* code length: supervisor */
437         EXTERN UWORD clos_label;        /* code length: OS label */
438         EXTERN UWORD clrtm;             /*  "     "   : real time monitor */
439         EXTERN UWORD clmem;             /*  "     "   : memory manager */
440         EXTERN UWORD clcio;             /*  "     "   : char i/o */
441         EXTERN UWORD clbdos;            /*  "     "   : bdos */
442         EXTERN UWORD clnet;             /*  "     "   : network handler */
443         EXTERN UWORD clxios;            /*  "     "   : xios */
444         EXTERN UWORD cltotal;           /*  "     "    of all modules */
445         EXTERN UWORD dsstart;           /* starting segment of system data */
446         EXTERN UWORD dlsysdat;          /* data length: system data */
447         EXTERN UWORD dlxios;            /* length of xios in data seg */
448         EXTERN UWORD dltables;          /* length of tables */
449         EXTERN UWORD dsotables;         /* starting offset of tables */
450         EXTERN UWORD dlrsps;            /* length of rsps */
451         EXTERN UWORD dltotal;           /* length of data, xios, tables & rsps */
452         EXTERN UWORD rsvd_seg;          /* start of OS reserved (buffer) area (not in load image) */
453         EXTERN UWORD bufs_seg;          /* ptr to allocated buffer at rsvd_seg */
454
455     /*** fixup locations ***/
456         EXTERN LONG fldstart;           /* location of SYSDAT start */
457         EXTERN LONG locmfl;             /* location in file of mem free list */
458
459     /*************************************************************/
460     /* system data variables for MP/M-86 or Concurrent CP/M-86 */
461         EXTERN ENTRY * sd_supmod;
462         EXTERN ENTRY * sd_rtmmod;
463         EXTERN ENTRY * sd_memmod;
464         EXTERN ENTRY * sd_ciomod;
465         EXTERN ENTRY * sd_bdosmod;
466         EXTERN ENTRY * sd_xiosmod;
467         EXTERN ENTRY * sd_netmod;
468         EXTERN UWORD * sd_mpmseg;
469         EXTERN UWORD * sd_rspseg;
470         EXTERN UWORD * sd_endseg;
471         EXTERN BYTE * sd_module_map;
472         EXTERN BYTE * sd_ncns;
473         EXTERN BYTE * sd_nlst;
474         EXTERN BYTE * sd_nccb;
475         EXTERN UBYTE * sd_nflags;
476         EXTERN BYTE * sd_srchdisk;
477         EXTERN UWORD * sd_mmp;
478         EXTERN BYTE * sd_nslaves;
479         EXTERN BYTE * sd_dayfile;
480         EXTERN BYTE * sd_tempdisk;
```

```
481         EXTERN BYTE * sd_tickspsec;
482         EXTERN UWORD * sd_lul;
483         EXTERN UWORD * sd_ccb;
484         EXTERN UWORD * sd_flags;
485         EXTERN UWORD * sd_mdul;
486         EXTERN BYTE * sd_nxmds;
487         EXTERN UWORD * sd_mfl;
488         EXTERN BYTE * sd_nmparts;
489         EXTERN UWORD * sd_pul;
490         EXTERN BYTE * sd_nxpd;
491         EXTERN UWORD * sd_qul;
492         EXTERN BYTE * sd_nqds;
493         EXTERN UWORD * sd_qmau;
494         EXTERN UWORD * sd_qmastart;         /* segment start of q buffer */
495         EXTERN UWORD * sd_qmalen;           /* length of q buffer */
496         EXTERN UWORD * sd_verptr;           /* pts to SUP segment ver string */
497         EXTERN UWORD * sd_vn;               /* F.12 version num */
498         EXTERN UWORD * sd_mpmvn;            /* F.163 version num */
499         EXTERN BYTE * sd_ncondev;
500         EXTERN BYTE * sd_nlstdev;
501         EXTERN BYTE * sd_nclodev;
502         EXTERN UWORD * sd_lcb;
503         EXTERN BYTE * sd_plock_max;
504         EXTERN BYTE * sd_popen_max;
505         EXTERN BYTE * sd_cmode;
506
507
508     #endif
509
510
511     #define PUTDRV(dd,nn) (sprintf(dd,"%c:%s","A"+destdrv,nn))
```



```
**** CROSS REFERENCE *****
      BDOSMOD  217, 235
        BREAD   15
       BROPEN   15
       BUFSIZ   72
      BUPDATE   16
      BWCREAT   16
          CCB  176, 177
         CCPM   29,  34, 205
       CIOMOD  216, 234
      CMDSLEN  248
        CRASH  244
       DBGPRT  247
        DLIST   76,  77,  79, 316, 317, 428, 429
        ENTRY  262, 263, 352, 353, 354, 355, 356, 357, 358, 461, 462, 463, 464,
               465, 466, 467
        ERROR   31,  36
       FCHECK  245
          FIX  271, 272, 276
         FLAG  170, 171
        F_ADD  278
        F_PUT  279
        GROUP   84,  85
     GTYPCODE   92
     GTYPDATA   93
         LOCK  148, 149
   MAINMODULE  289
```

```
        clsup  326, 436
      cltotal  334, 444
       clxios  333, 443
       cmdgrp   84
       cmdhdr  323, 434
      compile   36
    copyright  294, 406
        creat   16
           dd  511
      destdrv  304, 416, 511
     dliststr   76
       dlname   78
       dlnext   79
       dlrsps  340, 450
     dlsysdat  336, 446
     dltables  338, 448
      dltotal  341, 451
       dlxios  337, 447
      doclean  301, 413
     dogensys  300, 412
         doit  298, 410
   donew_xios  302, 414
    dsotables  339, 449
      dsstart  335, 445
   entrypoint  252
      ep_eoff  264
      ep_eseg  265
      ep_ioff  266
      ep_iseg  267
    exclusive   36
         exit  244
       f_addr  273
       f_next  276
        f_ptr  275
       f_type  274
      fixitem  271
        fl_pd  173
    fl_status  172
    flag_item  170
     fldstart  346, 456
           fn   15,  16, 244
          fns  299, 411
      fprintf  244
       gabase   88
         gets   19
         glen   87
         gmax   90
         gmin   89
        gtype   86
            h    8,   9
       ifndef   29,  30
      include    8,   9
      lo_link  150
   lo_restofit  151
     lock_item  148
        locmfl  347, 457
         lseek   17
        malloc   18
    md_length  126
```

CCP/M-86 2.0 V14
COPYRIGHT © 1981, 1982, 1983
DIGITAL RESEARCH
P. O. BOX 579
PACIFIC GROVE, CA 93950
SER. # CC-104

```
       version  295, 407
    xios_table  104
            xt  319, 431
      xt_alloc  116
        xt_ccb  113
       xt_door  108
     xt_dphtab  115
        xt_lcb  114
      xt_nccbs  111
      xt_nlcbs  112
      xt_nvcns  110
       xt_rsvd  109
       xt_tick  106
  xt_ticks_sec  107
       xtrapds  311, 423
           you   31
**** end cross reference ***
```

```
/*****************************************************************************
*
*           C P / M   C   R U N   T I M E   L I B   H E A D E R   F I L E
*           -------------------------------------------------------------
*       Copyright 1982 by Digital Research Inc.  All rights reserved.
*
*       This is an include file for assisting the user to write portable
*       programs for C.
*
*****************************************************************************/
#define UCHARA 1                                /* if char is unsigned      */
/*
 *      Standard type definitions
 */
                                                /****************************/
#define BYTE    char                            /* Signed byte              */
#define BOOLEAN int                             /* 2 valued (true/false)    */
#define WORD    int                             /* Signed word (16 bits)    */
#define UWORD   unsigned int                    /* unsigned word            */

#define LONG    long                            /* signed long (32 bits)    */
#define ULONG   long                            /* Unsigned long            */


#define REG     register                        /* register variable        */
#define LOCAL   auto                            /* Local var on 68000       */
#define EXTERN  extern                          /* External variable        */
#define MLOCAL  static                          /* Local to module          */
#define GLOBAL  /**/                            /* Global variable          */
#define VOID    /**/                            /* Void function return     */
#define DEFAULT int                             /* Default size             */
                                                /****************************/
#ifdef UCHARA
#define UBYTE   char                            /* Unsigned byte            */
#else
#define UBYTE   unsigned char                   /* Unsigned byte            */
#endif


/*****************************************************************************/
/*      Miscellaneous Definitions:                                           */
/*****************************************************************************/
#define FAILURE (-1)                    /*      Function failure return val */
#define SUCCESS (0)                     /*      Function success return val */
#define YES     1                       /*      "TRUE"                      */
#define NO      0                       /*      "FALSE"                     */
#define FOREVER for(;;)                 /*      Infinite loop declaration   */
#define NULL    0                       /*      Null pointer value          */
#define NULLPTR (char *) 0              /*                                  */
#define EOF     (-1)                    /*      EOF Value                   */
#define TRUE    (1)                     /*      Function TRUE  value        */
#define FALSE   (0)                     /*      Function FALSE value        */

/*************************** end of portab.h ********************************/
```

**** CROSS REFERENCE *****
**** end cross reference ***

GENCCPM.MAP

```
cs:686E $dadd
cs:6B97 $dceq
cs:6B80 $dcge
cs:6BAB $dcgr
cs:6BA6 $dcle
cs:6BA1 $dcls
cs:6B9C $dcne
cs:6929 $dcvti
cs:6924 $dcvtl
cs:695D $dcvtui
cs:6958 $dcvtul
cs:6ADF $ddiv
cs:6BFE $dload
cs:6A18 $dmul
cs:6815 $dneg
cs:6L0C $dstore
cs:6B66 $dsub
cs:6E18 $entry0
cs:6E1D $entry1
cs:6E30 $entry2
cs:6C22 $fload
cs:6C4A $fstore
cs:6C91 $lswitch
cs:6B23 $lcvtd
cs:6D46 $llshift
cs:6D57 $lmul
cs:6D7C $lrsshift
cs:6CB2 $lsdiv
cs:6CBA $lsmod
cs:6CB6 $ludiv
cs:6CBE $lumod
cs:6D8D $main
ds:231E _allocb
ds:2322 _allocd
cs:6E36 _checkfd
cs:5862 _errchk
cs:6DCD _exit
ds:1D08 _exittbc
cs:72D1 _fmtout
cs:57AD _fndcmd
cs:794E _main
cs:9163 _makefcb
cs:6067 _newdli
cs:571A _nxtcmd
cs:7BB4 _open
ds:22FE _opentab
cs:8003 _osread
cs:82CE _oswrite
cs:94ED _read
ds:1D06 _systype
ds:1C56 _sysuglo
ds:1CF4 _sysvals
cs:5E14 _wget
cs:98A4 _write
cs:394C _wrlbl
cs:64C3 access
cs:1F15 addmem
cs:44EF askabout
cs:4AB0 askhash
cs:4807 asknd
cs:49D3 askpm
cs:46B4 askstar
cs:8452 atoi
cs:5A87 atoih
cs:8561 bdos
cs:546C bldmenu
ds:20FD bufs_seg
cs:3BFF bufwrite
cs:856E calloc
cs:296E chkaddr
cs:041F chkmodul
ds:20E5 clbdos
ds:20E3 clcio
ds:1E0E clearit
ds:1C5C clearx
ds:20E1 clmem
ds:20E7 clnet
ds:20DD clos_lab
cs:85AD close
ds:20DF clrtm
ds:20DB clsup
ds:20EB cltotal
ds:20E9 clxios
ds:2058 cmdhdr
cs:2365 cntmlist
cs:0143 copyargs
ds:1C5A copynote
ds:1C5E copyrigh
cs:864C creat
cs:38D5 cseek
ds:22B6 csfd
ds:22B8 csgoff
cs:291E datwrite
cs:3F26 dbufs_di
cs:42DC dcalcsiz
cs:20E8 delmem
ds:1E10 destdrv
ds:22BC dh
ds:22DC dl
cs:5FA6 dirsearc
ds:20F7 dlrsps
ds:20CF dlsysdat
ds:20F3 dltables
ds:20F9 dltotal
ds:20F1 dlxios
ds:1C6A doclean
cs:0923 doexit
ds:1C68 dogensys
ds:1C64 doit
cs:381E dolbl
cs:1DF3 domem
cs:5511 domenu
ds:1C6C donew_xi
cs:14BB dorsp
cs:1428 dosysp
cs:3E99 doxbufs
cs:1395 doxios
ds:228A dphtab
ds:20F5 dsotable
cs:423C dspdnums
cs:40D2 dspdsize
cs:2267 dspmlist
ds:20ED dsstart
cs:0E2B edsys
ds:1C76 edsysver
cs:8667 exit
cs:86B9 fflush
cs:8761 fgets
cs:3A8F fixfile
cs:3A37 fixlater
ds:1CF0 fixroot
cs:0E8C fixups
cs:53CC fixupxio
ds:20FF fldstart
ds:1C66 fns
cs:8818 fopen
cs:8986 fprintf
cs:89AA fputs
cs:89EF free
cs:0D95 genfix
cs:3127 genmfl
cs:0A6B gensys
cs:29E1 gentable
cs:4C64 genxios
cs:65A3 getdrive
cs:6514 gets
cs:194C grpseek
cs:33A1 gwab_hdr
cs:1D70 help
ds:1CB4 helpml
```

```
ds:1CD8 helpm2
cs:8B28 index
cs:0597 init_eds
cs:119F init_sds
cs:0192 init_sys
cs:3C3E initxios
cs:2530 intomlis
cs:8B71 isatty
cs:8BAB isdigit
cs:8BF4 islower
cs:8C3D isspace
cs:8C97 isupper
ds:2103 locmfl
ds:22FC locrsvd_
cs:8CE0 lseek
cs:8EDF ltos
cs:0003 main
ds:1CAA mainmenu
ds:1C78 mainmtit
cs:09F0 make_sys
cs:9073 makefcb
cs:91F8 malloc
ds:1CAE memmenu
ds:1C74 memroot
cs:931D movmem
cs:58C5 msetbool
cs:598F msetbyte
cs:58DD msetdrv
cs:5B89 msettxt
cs:5A14 msetword
cs:1844 namematc
cs:2881 onbounds
cs:933F open
cs:2903 padwrite
cs:61F1 parsefn
cs:6615 patch
cs:65BC press_re
cs:935A printf
cs:5C84 prtmenu
cs:5CBE prtmival
ds:1C6E pure_xio
cs:3751 putname
ds:1E19 qbuflen
cs:937A read
cs:9616 rename
ds:2165 rhdr
ds:2167 rpd
cs:18D3 rsp_disp
cs:169D rspexclu
cs:1558 rspinclu
cs:1645 rspjoin
ds:1CB2 rspmenu
ds:1C70 rsps
cs:177B rspzap
ds:20FB rsvd_seg
ds:2169 ruda
cs:96B2 sbrk
ds:210F sd_bdosm
ds:2133 sd_ccb
ds:2100 sd_ciomo
ds:215F sd_cmode
ds:2128 sd_dayfl
ds:2119 sd_endse
ds:2135 sd_flags
ds:2159 sd_lcb
ds:2131 sd_lul
ds:2137 sd_mdul
ds:2108 sd_memmo
ds:2138 sd_mfl
ds:2127 sd_mmp
ds:211B sd_modul
ds:2115 sd_mpmse
ds:2151 sd_mpmvn
ds:2121 sd_nccb
ds:2157 sd_nciod
ds:211D sd_ncns
ds:2153 sd_ncond
ds:2113 sd_netmo
ds:2123 sd_nflag
ds:211F sd_nlst
ds:2155 sd_nlstd
ds:213D sd_nmpar
ds:2145 sd_nqds
ds:2129 sd_nslav
ds:2139 sd_nxmds
ds:2141 sd_nxpd
ds:2158 sd_plock
ds:215D sd_popen
ds:213F sd_pul
ds:2148 sd_qmale
ds:2149 sd_qmast
ds:2147 sd_qmau
ds:2143 sd_qul
ds:2117 sd_rspse
ds:2109 sd_rtmmo
ds:2125 sd_srchd
ds:2107 sd_supmo
ds:212D sd_tempd
ds:212F sd_ticks
ds:214D sd_verpt
ds:214F sd_vn
ds:2111 sd_xiosm
ds:1E4B stat_are
cs:28d1 segwrite
cs:9707 setaem
cs:9757 sprintf
cs:9777 strcat
cs:97D4 strcpy
cs:980F strlen
cs:9853 strncmp
cs:98E8 strncpy
ds:1C72 syslbls
ds:1CB0 syspmenu
cs:0938 sysvalid
cs:994B tolower
ds:1E11 totmds
ds:1E1B totopen
ds:1E13 totpds
ds:1E17 totqcbs
cs:9989 toupper
cs:2829 trimit
cs:259F trimlist
cs:99C7 unlink
ds:1C62 verbose
ds:1C60 version
cs:5EFB whichof
cs:109F wrapup
cs:9A20 write
cs:398D writelbl
cs:3682 xfer_mix
cs:3554 xfer_pur
cs:3280 xfer_rsp
cs:3440 xfer_sep
cs:1A97 xfergrp
cs:1AF6 xferpart
ds:1CAC xiosmenu
cs:4BAD xiosvali
ds:1E1D xt
ds:1E15 xtrapds
cs:3250 zfill
```