You can search this index. Type the keyword(s) you want to search for:

SLAC Phone and E-mail directory search.

Use standard SPIRES search terms such as...

```
   find name addis
   find name crane
<END>
```

You can search this index. Type the keyword(s) you want to search for:

SLAC Phone and E-mail directory search.

Supply Last-name or Last-name, First-name

```
/** eupdate: Do not modify this file (but use EUPDATE).
  * eupdate: It was built by the EUPDATE BUILD procedure.
  * eupdate: The source file was: BSDTYPES $H G1
  * eupdate: Built by jay - TCPMAINT at YKTVMZ
  * eupdate: Date: 05/22/90 - Time: 17:26:10 - EDT */
/*
 * 5799-WZQ (C) COPYRIGHT = NONE
 * LICENSED MATERIALS - PROPERTY OF IBM
 */
/* $Header:types.h 9.1$ */
/* $ACIS:types.h 9.1$ */
/* $Source: /ibm/acis/usr/sys/h/RCS/types.h,v $ */

#ifndef _TYPES_
#define _TYPES_

#if !defined(lint) && !defined(LOCORE) && defined(RCS_HDRS)
static char *rcsidtypes = "$Header:types.h 9.1$";
#endif

/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 *
 *  @(#)types.h    7.1 (Berkeley) 6/4/86
 */

/*
 * Basic system types and major/minor device constructing/busting\
 macros.
 */

/* major part of a device */
#define  major(x)  ((int)(((unsigned)(x)>>8)&0377))

/* minor part of a device */
#define  minor(x)  ((int)((x)&0377))

/* make a device number */
#define  makedev(x,y)   ((dev_t)(((x)<<8) | (y)))

typedef  unsigned char  u_char;
typedef  unsigned short u_short;
typedef  unsigned int   u_int;
typedef  unsigned long  u_long;
typedef  unsigned short ushort;        /* sys III compat */

#ifdef vax
typedef  struct    _physadr { int r[1]; } *physadr;
typedef  struct    label_t   {
    int  val[14];
} label_t;
#endif /* vax */
#if defined(ibm032) || defined(ibm370)
typedef  struct    _physadr { int r[1]; } *physadr;
typedef  struct    label_t   {
    int  val[11];
} label_t;
#endif /* defined(ibm032) || defined(ibm370) */
```

```
typedef  struct     _quad { long val[2]; } quad;
typedef  long daddr_t;
typedef  char *     caddr_t;
typedef  u_long     ino_t;
typedef  long swblk_t;
/* typedef  long size_t; terryh 10/15/91 */
typedef  long time_t; /* terryh 10/15/91 */
typedef  short      dev_t;
typedef  long off_t;
typedef  u_short    uid_t;
typedef  u_short    gid_t;
#ifdef __STDC__
typedef signed char prio_t;  /* priorities must be signed */
#else
typedef char prio_t;
#endif

#define  NBBY 8          /* number of bits in a byte */
/*
 * Select uses bit masks of file descriptors in longs.
 * These macros manipulate such bit fields (the filesystem macros use\
 chars).
 * FD_SETSIZE may be defined by the user, but the default here
 * should be >= NOFILE (param.h).
 */
#ifndef  FD_SETSIZE
#define  FD_SETSIZE      256
#endif

typedef long  fd_mask;
#define NFDBITS     (sizeof(fd_mask) * NBBY) /* bits per mask */
#ifndef howmany
#define  howmany(x, y)   (((x)+((y)-1))/(y))
#endif

typedef  struct fd_set {
    fd_mask    fds_bits[howmany(FD_SETSIZE, NFDBITS)];
} fd_set;

#define  FD_SET(n, p)    ((p)->fds_bits[(n)/NFDBITS] |= (1 << ((n) %\
 NFDBITS)))
#define  FD_CLR(n, p)    ((p)->fds_bits[(n)/NFDBITS] &= ~(1 << ((n) %\
 NFDBITS)))
#define  FD_ISSET(n, p) ((p)->fds_bits[(n)/NFDBITS] & (1 << ((n) %\
 NFDBITS)))
#define FD_ZERO(p) bzero((char *)(p), sizeof(*(p)))

#ifdef ibm032
/*
 * build a queue type appropriate to the particular machine
 * for historical reasons we use "caddr_t" on non-ibm032 machines.
 */
typedef struct qhdr {
    struct qhdr *link, *rlink;
} *queue_t;
#else /* ibm032 */
typedef caddr_t queue_t;
#endif /* ibm032 */
#endif
```

```
/* THIS EXEC FILE COMPILES AND LINKS A C PROGRAM */
/* gime c370 */
/* setup  (get tcpappl disk */
arg name
"GLOBAL TXTLIB EDCBASE IBMLIB CMSLIB"
/* "CC" name "(DEF(VM,IBM,DEBUG,SHORT_NAMES) xref list" */
   "CC" name "(DEF(VM,IBM,DEBUG,SHORT_NAMES)"
```

```
/* Compile all modules for WWW browser */

"cc2 www"
"cc2 htbuffer"
"cc2 htaccess"
"cc2 httcp"
"cc2 http"
"cc2 htftp"
"cc2 htparse"
"cc2 htfile"
"cc2 fudge"
"wload"
```

```
/* REXX */
/* trace r */
PARSE UPPER ARG fnList '(' opts
PARSE UPPER SOURCE . . . . . name .

ADDRESS COMMAND

if (fnList = '' | fnList = '?') then
  do
    'STATE' name 'HELPCMS *'
    if RC = 0 then
      do
        'HELP' name '(BRIEF'
        exit 6
      end
    else
      do
        'SET LANGUAGE (ADD EDC USER'
        'XMITMSG 001 (APPLID EDC NOHEADER'
        exit  10
      end
  end


/* SLAC Additions  */
'EXEC VMSTATUS PRESERVE'
/*'GLOBADD TXTLIB EDCBASE IBMLIB CMSLIB COMMTXT VSPASCAL AMPLANG' */
'EXEC GLOBADD TXTLIB EDCBASE IBMLIB'  /* terryh */
/*'EXEC GLOBADD LOADLIB EDCLINK'*/
'GLOBAL LOADLIB EDCLINK'
/* End - SLAC Additions */
comLib = 'IBMLIB'
remLib = 'PLILIB'  /* Must be removed from GLOBAL TXTLIB */
start = '(FROM CEESTART'
PARSE VAR fnList main fnList
reset= 'RESET CEESTART'
listLim = 0

'STATE EDCBASE TXTLIB *'
IF (rc ^= 0) THEN DO
  'SET LANGUAGE (ADD EDC USER'
  'XMITMSG 003 "EDCBASE" (APPLID EDC NOHEADER'
  'EXEC VMSTATUS RESTORE'
  EXIT 99
END

'STATE' comLib 'TXTLIB *'
IF (rc ^= 0) THEN DO
  'SET LANGUAGE (ADD EDC USER'
  'XMITMSG 003 comlib (APPLID EDC NOHEADER'
  'EXEC VMSTATUS RESTORE'
  EXIT 99
END

'STATE CMSLIB TXTLIB *'
IF (rc ^= 0) THEN DO
  'SET LANGUAGE (ADD EDC USER'
  'XMITMSG 003 "CMSLIB" (APPLID EDC NOHEADER'
  'EXEC VMSTATUS RESTORE'
  EXIT 99
```

```
END

'QUERY CMSTYPE (LIFO'
PULL cmstypeSav
PARSE UPPER VAR cmstypeSav junk1 '=' cmstypeSav


'QUERY TXTLIB (LIFO'
PARSE UPPER PULL 'TXTLIB' '=' txtList
IF txtList = 'NONE' THEN txtList= ''
txtListSav= txtList

i= FIND(txtList,'CMSLIB')
IF (i ^= 0) THEN DO
  txtList= DELWORD(txtList,i,1)
END

i= FIND(txtList,'EDCBASE')
IF (i ^= 0) THEN DO
  txtList= DELWORD(txtList,i,1)
END

i= FIND(txtList,comLib)
IF (i ^= 0) THEN DO
  txtList= DELWORD(txtList,i,1)
END

fnList = space(fnList)

'GLOBAL TXTLIB EDCBASE ' comLib 'CMSLIB' txtList

tempName = '$$$L$$D $$$M$P$ A1'


/* set load option defaults */
loadOpts = 'DUP NOAUTO INV MAP RLDSAVE'
vmxaOpts = ''
renameLoadMap = 0
modname = main
strinit = 'STR'
origin = ''
numopts = words(opts)
i = 1
DO WHILE(i <= numopts)
  opt = word(opts, i)
  SELECT
    WHEN opt = 'MODNAME' THEN DO
      modName = WORD(opts,i+1)
      i = i + 1
    END
    WHEN opt = 'ORIGIN' THEN DO
      origin = 'ORIGIN' WORD(opts,i+1)
      i = i + 1
    END
    WHEN opt = 'AUTO' THEN
      loadOpts = loadOpts 'AUTO'
    WHEN opt = 'NODUP' THEN
      loadOpts = loadOpts 'NODUP'
    WHEN opt = 'NOINV' THEN
      loadOpts = loadOpts 'NOINV'
```

```
      WHEN abbrev('NORLDSAVE', opt, 5) THEN DO
        k = FIND(loadOpts, 'RLDSAVE')
        IF (k ^= 0) THEN
          loadOpts = DELWORD(loadOpts,k,1)
      END
      WHEN opt = 'NOSTR' THEN
        strinit = 'NOSTR'
      WHEN opt = 'STR' then
        strinit = 'STR'
      WHEN opt = 'NOMAP' THEN DO
        loadOpts = loadOpts 'NOMAP'
        'STATE LOAD MAP A'
        IF (rc = 0) THEN DO
          'ERASE ' tempname
          'RENAME LOAD MAP A 'tempName
          renameLoadMap = 1
        END
      END
      WHEN opt = 'RMODE' | opt = 'AMODE' THEN DO
        vmxaOpts = vmxaOpts opt WORD(opts,i+1)
        i = i + 1
      END
      WHEN opt = 'MAP' | opt = 'NOAUTO' | opt = 'INV' |,
           opt = 'DUP' |,
           abbrev('RLDSAVE',opt, 3) THEN
        loadOpts = loadOpts opt
      OTHERWISE
        'SET LANGUAGE (ADD EDC USER'
        'XMITMSG 004 opt (APPLID EDC NOHEADER'
    END
    i = i + 1
  END


  IF (LENGTH(fnList) > listLim) THEN DO
      /* cut it up */
    i = 1
    DO WHILE (LENGTH(fnList) > 300)
      break = WORDS(left(fnList,200)) - 1
      fnl.i = SUBWORD(fnList,1,break)
      i = i + 1
      fnList = SUBWORD(fnList,break+1)
    END
    fnl.i = fnList
    fnl.0 = i - 1
    'SET CMSTYPE HT' /* terryh */
     /* DO the initial load */
    'LOAD' main '(CLEAR' origin loadOpts vmxaOpts reset
    IF (RC > 4) THEN DO
      'SET CMSTYPE RT'
      'LOAD' main '(CLEAR' origin loadOpts vmxaOpts reset
    END
    ELSE DO
      DO k = 1 TO fnl.0
        'INCLUDE  'fnl.k  '(NOCLEAR ' loadOpts reset
      END
      'SET CMSTYPE RT'
      'INCLUDE  'fnl.i  '(NOCLEAR ' loadOpts reset
    END
  END
```

```
ELSE DO
  'LOAD' main fnList '(CLEAR' origin loadOpts vmxaOpts reset
END

saveRc= rc
'GLOBAL TXTLIB 'txtListSav
'SET CMSTYPE 'cmstypeSav

IF (renameLoadMap) THEN DO
  'ERASE LOAD MAP A'
  'RENAME ' tempname 'LOAD MAP A'
END

IF (saveRc ^= 0) THEN DO
  'SET LANGUAGE (ADD EDC USER'
  'XMITMSG 005 (APPLID EDC NOHEADER'
  'EXEC VMSTATUS RESTORE'
  EXIT saveRc
END
'GENMOD' modName start strinit
'EXEC VMSTATUS RESTORE'
EXIT (rc)
```

```
DISK:      A 191  A   RW TYH193 1   3390 4096 -        -
DISK:      B 192  B   RW TYH    3   3390 2048 -        -
DISK:      C 193  C   RW TYH191 5   3390 1024 -        -
DISK:      D 194  D   RW TYH194 4   3390 4096 -        -
DISK:      E 195  E   RO H-EXEC 5   3390 2048 -        -
DISK:      F 196  F   RO IBM-C  17  3390 4096 -        -
DISK:      G 2EC  G   RO ECP192 220 3390 4096 -        -
DISK:      H 198  H   RO MNT592 43  3390 4096 -        -
DISK:      I -
DISK:      J -
DISK:      K -
DISK:      L -
DISK:      M -
DISK:      N -
DISK:      O -
DISK:      P -
DISK:      Q -
DISK:      R -
DISK:      S 190  S   RO MNT190 32  3390 4096 -        -
DISK:      T 59E  T   RO CMS59E 200 3390 4096 -        -
DISK:      U 59F  U   RO CMS59F 150 3390 4096 -        -
DISK:      V -
DISK:      W 5A1  W   RO CMS5A1 17  3390 4096 -        -
DISK:      X -
DISK:      Y 19E  Y/S RO CMS19E 200 3390 4096 -        -
DISK:      Z -
DOSLIBS:   -
LOADLIBS:  VSF2LOAD EDCLINK
MACLIBS:   -
TXTLIBS:   IBMLIB   CMSLIB   EDCBASE  COMMTXT  VSPASCAL AMPLANG
```

CPLINK: Writable Static Map 10/16/91 18:46.36

INFORMATIONAL EDC0920: No map displayed as no writable static was found.

SLACVM Information Service

BINLIST
: SLAC phone book with e-mail addresses

HEP
: SPIRES HEP preprint database

SLACVM Information Service

BINLIST
: SLAC phone book with e-mail addresses

HEP
: SPIRES HEP preprint database

SLACVM Information Service

BINLIST
: SLAC phone book with e-mail addresses

HEP
: SPIRES HEP preprint database

STORES
: SLAC Stores Catalog

```
/*  LOAD the DAEMON mand text files */
trace r
  "GLOBAL TXTLIB IBMLIB CMSLIB EDCBASE COMMTXT VSPASCAL AMPLANG"
"CMOD HTDAEMON FINDGATE HTTCP (nomap"
```

```
edcxv(argc)
  int argc;
{
  printf("here here");
}
```

```
/* FIND server version, June 1991                                    */
/* arg arg
   say "FSEARCH: received ->" arg
   queue "FSEARCH: received ->" arg
   exit 1
*/
 address command; arg host inp; rc=0
 Say '...FSEARCH  Start->'inp'.................'
 parse var inp k_ '(' disks
 if substr(inp,1,1)='!' Then CALL SPECIAL
'GLOBALV SELECT SPECIAL GET TRC'; If trc='I'Then trace i;else trace off
/*_____________________ Initialization ____________________________*/
 null=d2c(0)
 If disks='' Then disks='PUB USER'       /* Later done by client     */
 indsk='Z'                               /* OK for single index disk */
 parse var disks d_1 .                   /* To distinguish NEWS/non-N */
 k_=translate(k_,'O','0')                /* Zeroes and Os alike      */
 k_=translate(k_,' ','.-')               /* Suppress dots, hyphens   */
 If pos('(',disks) > 0 Then Do           /* Cope with NEWS: start    */
  parse var disks _.1 '('cats.1')'_.2'(' cats.2 ')'_.3'('cats.3'       )'
  ng=0
  Do k=1 to 3; If _.k=''Then leave
    do kk=1 to words(cats.k)
      ng=ng+1;disk.ng=strip(_.k); k_.ng=':'word(cats.k,kk)' 'k_
    end kk
  End
 End /* of '(' inside disks              / Cope with NEWS: end      */
 Else Do /*                              / Normal disk list: start  */
   ng=words(disks)
  'PIPE var disks | split | stem disk.'
   k_.=k_
 End /*                                  / Normal disk list: end    */
 If k_.1='' Then Call OUT 1 'You must give at least one keyword'
 nk=words(k_.1)
/*_____________ Getting all the exp-line numbers ___________________*/
kanz.=0; ktot=0; numeric digits 40
Do ig=1 to ng; g=disk.ig; lm=99999
  If ^FEXIST('FIND$KEY 'g indsk) Then Call OUT 8 "FIND group: '"g"' unknown"
  If QFILE('FIND$KEY 'g indsk,'RECNO')<10000 Then hsh=4999;Else hsh=49999
  Do k=1 to nk
    parse value word(k_.ig,k) with y 13;h=c2d(y)//hsh+3
   'EXECIO 1 DISKR FIND$KEY 'g indsk h' (VAR T'
    parse var t ys''rest; n=find(ys,y)
    If n=0 Then do; lm=0;leave; end
    Else Do
      nn=substr(rest,n*2-1,4);parse var nn n1 3 n2 5
      n1=c2d(n1);n2=c2d(n2)-n1;a.k=substr(rest,n1,n2)
      if lm>n2 Then do; lm=n2; km=k; end
    End
  End k
  Do k=2 to lm by 2;parse var a.km t 3 a.km
   Do i=1 to nk;if i=km Then iterate
     g=pos(t,a.i); If g=0 Then leave
     If g//2^=1 Then do
       s=substr(a.i,g-1)
       if s>t then do;g=0;leave;end
       a.i=substr(a.i,g+1);i=i-1
     End
     Else a.i=substr(a.i,g+2)
   End i;if g=0 & nk>1 Then Iterate
```

```
    ktot=ktot+1; kanz.ig=kanz.ig+1;kanz=kanz.ig;b.ig.kanz=c2d(t)
   End k
 End ig
 If ktot=0 Then Call OUT 4 k_' not found in the FIND groups: 'disks
 numeric digits
 /*._____________ Turning line numbers into exp-lines _____________*/
 nw=0
 Do ig=1 to ng; kanz=kanz.ig; g=disk.ig
   lastl.='';ln.0=null; lastl.0=null
   Do kk=1 to kanz
    'EXECIO 1 DISKR FIND$EXP 'g indsk b.ig.kk '(VAR _'
     If rc>0 Then call x_ rc
     parse var _ _(null)levels
     parse var levels lv 2 ln.1 4 ln.2 6 ln.3 8 ln.4 10; lv=c2d(lv)-1
     If lastl.lv=ln.lv Then Do; nw=nw+1; out.nw=_; end
     Else Do llv=lv-1 to 0 by -1
       If lastl.llv=ln.llv Then Do
        Do lllv=llv+1 to lv
          line=c2d(ln.lllv); lastl.lllv=ln.lllv
         'EXECIO 1 DISKR FIND$EXP 'g indsk line '(VAR __'
          If rc>0 Then call x_ rc
          nw=nw+1;parse var __ out.nw (null) .
        End lllv
        nw=nw+1; out.nw=_
        leave
       End
     End llv
   End kk
End ig
CALL OUT 0

/*--------------------------------------------------------------------*/
 OUT: arg rc text

 If rc=0 Then Do ktrim=1 to nw
   If pos('<A HREF',out.ktrim) > 0 Then Do
     parse var out.ktrim _1 '<A HREF=' _2  '>' _3 _4
     _3=translate(_3,' ','-')
     parse var _3 _31 _32
     If datatype(_3,'W') Then _3=right(_3,2)
     queue _1||'<A HREF=/FIND/'||translate(_2,'+',' ')||'>'
     queue _3'  '_4
   End
   Else queue out.ktrim
 End ktrim
 Else Do; ktot=-rc; If text^='' Then queue text; End

 If d_1='NEWS' Then monitor='XNEWSMON'; Else monitor = 'FINDMON'
'EXECIO 0 CP (STRING SMSG UMON !LOG! 'monitor right(ktot,4) inp
 Say '   FSEARCH  End  -> rc='right(rc,4)' stacksize='right(queued(),4)
 If pos('@',host) > 0 Then parse var host user '@' h1'.'h2'.'h3'.'h4 .
 Else parse var host h1'.'h2'.'h3'.'h4 .
 host=d2x(h1,2)||d2x(h2,2)||d2x(h3,2)||d2x(h4,2)
 monstring=left(host,9)||substr(date(),1,6)' 'substr(time(),1,5)
 monstring=monstring||right(queued(),5)' 'inp
'EXECIO 1 DISKW FSEARCH LOGSHEET C (VAR MONSTRING'
 Exit queued()

/*====================================================================*/
SPECIAL:
```

```
parse var k_ _1 _2 _3
If _1=!SPECIAL_BERND! Then Do
 'EXEC SPECIAL '_2 _3; Call OUT 99
End
Return 0
```

```
/* */
Parse arg ip_adr file '(' options
Say 'FGET received:' ip_adr file '(' options
Trace R
Parse var file fn '.' ft .

If ft = '' then do  /* attempt to initiate keyword search */
    'EXECIO * DISKR' fn 'INDEX ( FINI'
    Exit Queued()
    End /* do */

If ft /= 'html' then do  /* attempt to fetch file */
    Queue '<PLAINTEXT>'
    End /* do */
'EXECIO * DISKR' fn ft '( FINI'
Exit Queued()
```

```
/* */
Parse upper arg ip_adr file '(' options
Say time()' FGET received:' ip_adr file '(' options
/*
Address SPIRES "if $SELECT ^='HEP' Then :Sel HEP//thr binlist sel binlist'"
*/
Parse var file fn '.' ft .

If ft = '' then do  /* attempt to initiate keyword search */
    'EXECIO * DISKR' fn 'INDEX ( FINI'
    Exit Queued()
    End /* do */

if fn = 'DEFAULT' & ft = 'HTML' Then Do
    /* if this is a slac ip addr then get special defaults */
    If left(ip_adr,7) = '134.79.' Then Do
        fn = 'DEFAULTX'
        Say '.. SLAC user switching to 'fn' HTML'
        End
    End

If ft /= 'HTML' then do  /* attempt to fetch file */
    Queue '<PLAINTEXT>'
    End /* do */

'EXECIO * DISKR' fn ft '( FINI'
Exit Queued()
```

```
/* */
arg arg
say  "->" arg
exit
ADDRESS command  /* PAD 3 before each PIPE STACK is a "fix" for the */
arg host iinnpp  /* problem with ?? being taken as EOF by IBM C     */
 If pos('@',host) > 0 Then parse var host user '@' h1'.'h2'.'h3'.'h4 .
 Else parse var host h1'.'h2'.'h3'.'h4 .
 host=d2x(h1,2)||d2x(h2,2)||d2x(h3,2)||d2x(h4,2)
 monstring=left(host,9)||substr(date(),1,6)' 'substr(time(),1,5)
Say '   NICFOUN1 Start->'iinnpp
parse var iinnpp fid '('dispmode'/'glued'/'versions')'
versions=translate(versions,' ','+')
dispmode=translate(dispmode,' ','+')
Queue '<SEEINDEX HREF=/FIND>'
If fid='' Then do; queue 'No input whatsoever'; call get_out; end
parse var fid group '.'diskname'.'ft'.'fn .
 monstring=monstring'  DOC'left(group,7),
  left(diskname,8)||left(ft,8)||left(fn,8)
Say '   mode='dispmode 'glued='glued' versions='versions
SELECT /*_____________________________________________________________*/
 WHEN datatype(dispmode,'X') THEN DO
   CALL GET_MODE group'.'diskname 0;m_=result
   queue '<PLAINTEXT>'
   parse var dispmode _1 5 _2 7
   _1=x2d(_1)-1; _2=x2d(_2)
  'PIPE < 'fn ft m_' | drop '_1' | take '_2' | PAD 3 | STACK'
 END /* dispmode is a heXadecimal number --> linenumbers='X' */
 WHEN dispmode='X' THEN DO
   Call GET_MODE group'.'diskname 0;m_=result
   If glued='A' Then do
     pft='PRO:'||substr(ft,5)
     CALL GET_MODE group'.P_'diskname 0;m_p=result
   End; Else do;m_p=m_;pft=ft;end
  'EXECIO 2 DISKR 'fn pft m_p' 2 (STEM _. MAR 9 * '
   title=translate(_.1,' ','-')' 'space(_.2,1)
   queue '<title>'title'</title>'
   queue '<PLAINTEXT>'
   If ft='HELPPROG' Then 'PIPE < 'fn ft m_,
                         '| NFIND . | SPEC 12-90 1 | PAD 3 | STACK'
   Else 'PIPE < 'fn ft m_' | NFIND . | PAD 3 | STACK'
END /* of dispmode='X' */
WHEN dispmode='W' THEN DO
  pft='PRO:'||substr(ft,5)
  CALL GET_MODE group'.P_'diskname 0;m_p=result
 'EXECIO 2 DISKR 'fn pft m_p' 2 (STEM _. MAR 9 * '
  title=translate(_.1,' ','-')' 'space(_.2,1)
  queue '<title>'title'</title>'
 'EXECIO * DISKR 'fn pft m_p' (STEM _. MAR 9 * FIND /.cm TYP:/'
  If rc>0 Then types=''; else types=_.1
  Select
   When find(versions,'H') >0 Then do
     queue '<PLAINTEXT>'
     Call GET_MODE group'.'diskname 0;m_=result
    'PIPE < 'fn ft m_' | PAD 3 | STACK'
   End
   When find(versions,'S') >0 Then do
     parse var types 'SOURCE='sft . .
     If sft^='' Then do
      Call GET_MODE group'.S_'diskname 0;m_=result
```

```
        'PIPE < 'fn sft m_' | PAD 3 | STACK'
        End
        Else queue 'Something wrong with source'
      End
      Otherwise queue 'No displayable version available'
     End
   END /* of dispmode='W' */
   WHEN dispmode='H' THEN DO
     Call GET_MODE group'.'diskname 0;m_=result
     If glued='A' Then do
       pft='PRO:'||substr(ft,5)
       CALL GET_MODE group'.P_'diskname 0;m_p=result
     End; Else do;m_p=m_;pft=ft;end
    'EXECIO 2 DISKR 'fn pft m_p' 2 (STEM _. MAR 9 * '
     title=translate(_.1,' ','-')' 'space(_.2,1)
     queue '<title>'title'</title>'
     queue '<PLAINTEXT>'
     If ft='HELPPROG' Then 'PIPE < 'fn ft m_,
                           ' | NFIND . | SPEC 12-90 1 | PAD 3 | STACK'
     Else 'PIPE < 'fn ft m_' | NFIND . | PAD 3 | STACK'
   END /* of dispmode='H' */
   OTHERWISE queue 'Something is wrong, tell BERND@CERNVM'
  END /*_____________________________________________________________*/
  /*  'PIPE Stack | > bla bla a | console' */
  CALL GET_OUT
  RETURN queued()
  /*********************************************************************/
  GET_MODE:  procedure; arg disk flag
  'GLOBALV SELECT :XFIND_0 GET ACC_DISKS '
   parse var acc_disks acc_disks ',' acc_links ',' acc_modes
   nn_n=find(acc_disks,disk)
   if nn_n>0 Then RETURN subword(acc_modes,nn_n,1)
  'GLOBALV SELECT :XFIND_1 GET 'disk
   parse value value(disk) with id addr uaddr umode rpw .
   if id='' then do
      Queue 'The disk 'disk' is not defined'
      Call Get_out
   end
   If length(umode)=1 Then flag=1  /* disk accessed for all modes       */
   If flag & umode ^='.' & umode ^='' then RETURN umode
   else do
      l_=qdisk('?','ADDRESS')
     'EXECIO 1 CP (SKIP STRING LINK TO 'id addr 'AS' l_ 'RR' rpw
      If rc>2 Then Do
        Queue 'Linking problem with disk 'disk': TELL BERND@CERNVM'
        Call Get_out
      End
      zz=typemode('HT');m_=QDISK('?','MODE','-Z');'ACCESS 'l_ m_;zz=typemode(zz)
      If rc>0 Then Do
        Queue 'Accessing problem with disk 'disk': TELL BERND@CERNVM'
        Call Get_out
      End
      acc_disks=acc_disks' 'disk
      acc_links=acc_links' 'l_
      acc_modes=acc_modes' 'm_
     'GLOBALV SELECT :XFIND_0 SETL ACC_DISKS 'acc_disks','acc_links','acc_modes
   end
  RETURN m_
  /*-------------------------------------------------------------------*/
  GET_OUT: monstring=monstring' 'title
```

```
'EXECIO 1 DISKW FGET    LOGSHEET C (VAR MONSTRING'
'GLOBALV SELECT :XFIND_0 GET ACC_DISKS '
 parse var acc_disks acc_disks ',' acc_links ',' acc_modes
'SET CMSTYPE HT'
 do k=1 to words(acc_links)
  'RELEASE 'WORD(acc_modes,k)
  'EXECIO 1 CP (SKIP STRING DETACH 'WORD(acc_links,k)
 end
'SET CMSTYPE RT'
'GLOBALV SELECT :XFIND_0 PURGE'
Say '   NICFOUN1 End  -> stacksize='queued()
EXIT queued()
```

```
/*              Handle HTTP request for CERNVM FIND server      FINDGate.c
**
**      This code takes a document name and converts it into
**      a VM CMS command to produce the hypertext the caller requires.
**      This involves, in general, calling an EXEC file which puts
**      the data onto the "stack" The stack allows the data to be read from
**      stdin as though it came from the command device.
**      The commands are written to return a value which is equal to the
**      number of lines put onto the stack.
**
** History
**      29 May 91       Separated from daemon program - TBL
**      19 May 91       Retrieve subroutine format used. -TBL
*/

#define BUFFER_SIZE 4096        /* Arbitrary size for efficiency */

#include "HTUtils.h"
#include "tcp.h"                /* The whole mess of include files */
#include "HTTCP.h"              /* Some utilities for TCP */

extern int WWW_TraceFlag;       /* Control diagnostic output */
extern FILE * LogFile;          /* Log file output */
extern char HTClientHost[16];   /* Client name to be output */
extern int HTWriteASCII(int soc, const char * s);       /* In HTDaemon.c */



/*      Strip white space off a string
**
** On exit,
**      Return value points to first non-white character, or to 0 if none.
**      All trailing white space is overwritten with zero.
*/
PRIVATE char * strip(char * s)
{
#define SPACE(c) ((c==' ')||(c=='\t')||(c=='\n'))
    char * p=s;
    for(p=s;*p;p++);                    /* Find end of string */
    for(p--;p>=s;p--) {
        if(SPACE(*p)) *p=0;     /* Zap trailing blanks */
        else break;
    }
    while(SPACE(*s))s++;        /* Strip leading blanks */
    return s;
}



/*      Handle one message
**      ------------------
**
** On entry,
**      soc             A file descriptor for input and output.
** On exit,
**      returns         >0      Channel is still open.
**                      0       End of file was found, please close file
**                      <0      Error found, please close file.
*/
/*      Retrieve information
```

```
**      --------------------
*/
#ifdef __STDC__
int HTRetrieve(const char * arg, const char * keys, int soc)
#else
int HTRetrieve(arg, keys, soc)
    char *arg;
    char *keys;
    int soc;
#endif
{
#define COMMAND_SIZE    255
#define MAX_LINES 10000                 /* Maximum number of lines returned */
    char command[COMMAND_SIZE+1];
    char keywords[COMMAND_SIZE+1];
    char argument[COMMAND_SIZE+1];
    char buffer[BUFFER_SIZE];
    int status;
    char * filename;    /* Pointer to filename or group list*/
    char system_command[COMMAND_SIZE+1];
    int lines;          /* Number of lines returned by EXEC file */
    int fd;             /* File descriptor number */
    int isIndex = 0;    /* Is the thing asked for an index? */

/*      Remove host and any punctuation. (Could use HTParse @)
*/
    strcpy(argument, arg);
    filename = argument;
    if (argument[0]=='/') {
        if (argument[1]=='/') {
            filename = strchr(argument+2, '/'); /* Skip //host/ */
            if (!filename) filename=argument+strlen(argument);
        } else {
            filename = argument+1;      /* Assume root: skip slash */
        }
    }

    if (!*filename) {
        HTWriteASCII(soc,"<P>Error: No document ID provided.<P>\n");
        return 0;

    }

    if (keys) strcpy(keywords, keys);
    else keywords[0] = 0; /* As if just "?" given (historical) */

/*      Formats supported by FIND are:
**
**      /FIND                           Help for find general index
**      /FIND/                          (same)
**      /FIND/?                         (same)
**      /FIND/group.disk.ft.fn          Get a document from a disk- NOT INDEX
**      /FIND/grouplist?                Help for group index
**      /FIND/grouplist?keylist         Search a set of disks for keywords
**                                      where grouplist = group+group+group.. or voi
**                                      keylist = key+key+key... (not void)
*/
    if((
            (0==strncmp(filename,"FIND",4))
            || (0==strncmp(filename,"find",4))
```

```
        ) && (
             (filename[4]==0)
             || (filename[4]=='/')
             || (filename[4]=='?')
        )) {

        filename=filename+4;
        if (*filename=='/') filename++; /* Skip first slash */

        if (keys || !*filename) {                    /* Index access */
            char *p;
            while((p=strchr(filename,'+'))!=0) *p=' ';/* Remove +s */
            while((p=strchr(keywords,'+'))!=0) *p=' ';/* Remove +s */

            if (!*keywords) {                        /* If no keywords    */
                HTWriteASCII(soc, "<IsIndex><TITLE>");
                HTWriteASCII(soc, filename);
                HTWriteASCII(soc, " keyword index</TITLE>\n");
                if (*filename) {
                    HTWriteASCII(soc, "This index covers groups: `");
                    HTWriteASCII(soc, filename);
                    HTWriteASCII(soc, "'");
                } else {
                    HTWriteASCII(soc,
"This is the general CERN Computer Center public document index");
                }
                HTWriteASCII(soc,
". Your words will be matched against author-supplied keywords.\n");
                HTWriteASCII(soc,
"<P>Supply keywords to search for information.<P>\n");
                return 0;
            }

            sprintf(buffer, "<IsIndex>\n<TITLE>%s in %s</TITLE>\n",
                keywords, *filename ? filename : "general index" );
            HTWriteASCII(soc, buffer);

            sprintf(system_command,"EXEC FSEARCH %s %s",
                HTClientHost, keywords);
            if (*filename) {
                strcat(system_command, " ( ");
                strcat(system_command, filename);
            }
        } else {
            sprintf(system_command,"EXEC FGET %s %s",
                HTClientHost, filename);
        }

    }  /* if FIND */

/*      Formats supported by NEWS are:
**
**      /NEWS                               General news
**      /NEWS/grouplist                     News for given groups
**      /NEWS/grouplist?                    All news for all given groups
**      /NEWS/grouplist?keylist             Search news for keywords
**                                          where grouplist = group+group+group.. or voi
**                                          keylist = key+key+key... (can be void)
*/
    else if((
```

```
                (0==strncmp(filename,"NEWS",4))
                || (0==strncmp(filename,"news",4))
            ) && (
                (filename[4]==0)
                || (filename[4]=='/')
                || (filename[4]=='?')
            )) {

        filename=filename+4;
        if (*filename=='/') filename++;          /* Skip first slash */

        {
            char *p;
            while((p=strchr(filename,'+'))!=0) *p=' ';/* Remove +s */
            while((p=strchr(keywords,'+'))!=0) *p=' ';/* Remove +s */

            sprintf(system_command,"EXEC FSEARCH %s", HTClientHost);
            if (keywords) strcat(system_command, keywords);
            strcat(system_command, " ( NEWS ");
            if (*filename) {
                strcat(system_command, "(");
                strcat(system_command, filename);
                strcat(system_command, ") ");
            }
        }

        sprintf(buffer, "<IsIndex>\n<TITLE>%s NEWS in %s</TITLE>/n",
            keywords, filename ? filename : "general index" );
        HTWriteASCII(soc, buffer);

/*      Pick up other unknown forms
*/
    } else { /* not FIND or NEWS */
        HTWriteASCII(soc, "Bad syntax. No such document.\n");
        return 0;
    }

/*      Execute system command and get the results
*/
    lines = system(system_command);      /* Number of stacked lines */
    if (TRACE) printf("Command `%s' returned %i lines.\n",
                system_command, lines);
    if (lines<=0) {
        HTWriteASCII(soc,
        "\nSorry, the FIND server could not execute\n  `");
        HTWriteASCII(soc, system_command);
        HTWriteASCII(soc, "'\n\n");
        return 0;
    }

    /*  Copy the file across: */

    for(;lines; lines--){                /* Speed necessary here */
        char *p;
        if (!fgets(buffer, BUFFER_SIZE, stdin))
            sprintf(buffer,
"*** End of file with %d lines left after command\n%s\n - mail BERND@CERNVM",
            lines, system_command);
        for(p=buffer; *p; p++) *p = TOASCII(*p);
        write(soc, buffer, p-buffer);
```

```
    }
    return 0;

} /* HTRetrieve() */
```

```
/*              Handle HTTP request for CERNVM FIND server      FINDGate.c
**
**      This code takes a document name and converts it into
**      a VM CMS command to produce the hypertext the caller requires.
**      This involves, in general, calling an EXEC file which puts
**      the data onto the "stack" The stack allows the data to be read from
**      stdin as though it came from the command device.
**      The commands are written to return a value which is equal to the
**      number of lines put onto the stack.
**
** History
**      29 May 91       Separated from daemon program - TBL
**      19 May 91       Retrieve subroutine format used. -TBL
*/

#define BUFFER_SIZE 4096        /* Arbitrary size for efficiency */

#include "HTUtils.h"
#include "tcp.h"                /* The whole mess of include files */
#include "HTTCP.h"              /* Some utilities for TCP */

extern int WWW_TraceFlag;       /* Control diagnostic output */
extern FILE * LogFile;          /* Log file output */
extern char HTClientHost[16];   /* Client name to be output */
extern int HTWriteASCII(int soc, const char * s);       /* In HTDaemon.c */



/*      Strip white space off a string
**
** On exit,
**      Return value points to first non-white character, or to 0 if none.
**      All trailing white space is overwritten with zero.
*/
PRIVATE char * strip(char * s)
{
#define SPACE(c) ((c==' ')||(c=='\t')||(c=='\n'))
    char * p=s;
    for(p=s;*p;p++);                    /* Find end of string */
    for(p--;p>=s;p--) {
        if(SPACE(*p)) *p=0;     /* Zap trailing blanks */
        else break;
    }
    while(SPACE(*s))s++;        /* Strip leading blanks */
    return s;
}



/*      Handle one message
**      ------------------
**
** On entry,
**      soc             A file descriptor for input and output.
** On exit,
**      returns         >0      Channel is still open.
**                      0       End of file was found, please close file
**                      <0      Error found, please close file.
*/
/*      Retrieve information
```

```
**      --------------------
*/
#ifdef __STDC__
int HTRetrieve(const char * arg, const char * keys, int soc)
#else
int HTRetrieve(arg, keys, soc)
    char *arg;
    char *keys;
    int soc;
#endif
{
#define COMMAND_SIZE    255
#define MAX_LINES 10000                     /* Maximum number of lines returned */
    char command[COMMAND_SIZE+1];
    char keywords[COMMAND_SIZE+1];
    char argument[COMMAND_SIZE+1];
    char buffer[BUFFER_SIZE];
    int status;
    char * filename;    /* Pointer to filename or group list*/
    char system_command[COMMAND_SIZE+1];
    int lines;          /* Number of lines returned by EXEC file */
    int fd;             /* File descriptor number */
    int isIndex = 0;    /* Is the thing asked for an index? */

/*      Remove host and any punctuation. (Could use HTParse @)
*/
    strcpy(argument, arg);
    filename = argument;
    if (argument[0]=='/') {
        if (argument[1]=='/') {
            filename = strchr(argument+2, '/'); /* Skip //host/ */
            if (!filename) filename=argument+strlen(argument);
        } else {
            filename = argument+1;      /* Assume root: skip slash */
        }
    }

    if (!*filename) {
        HTWriteASCII(soc,"<P>Error: No document ID provided.<P>\n");
        return 0;

    }

    if (keys) strcpy(keywords, keys);
    else keywords[0] = 0; /* As if just "?" given (historical) */

/*      Formats supported by FIND are:
**
**      /FIND                           Help for find general index
**      /FIND/                          (same)
**      /FIND/?                         (same)
**      /FIND/group.disk.ft.fn          Get a document from a disk- NOT INDEX
**      /FIND/grouplist?                Help for group index
**      /FIND/grouplist?keylist         Search a set of disks for keywords
**                                      where grouplist = group+group+group.. or voi
**                                      keylist = key+key+key... (not void)
*/
    if((
            (0==strncmp(filename,"FIND",4))
            || (0==strncmp(filename,"find",4))
```

```
        ) && (
            (filename[4]==0)
            || (filename[4]=='/')
            || (filename[4]=='?')
        )) {

        filename=filename+4;
        if (*filename=='/') filename++; /* Skip first slash */

        if (keys || !*filename) {                /* Index access */
            char *p;
            while((p=strchr(filename,'+'))!=0) *p=' ';/* Remove +s */
            while((p=strchr(keywords,'+'))!=0) *p=' ';/* Remove +s */

            if (!*keywords) {                    /* If no keywords    */
                HTWriteASCII(soc, "<IsIndex><TITLE>");
                HTWriteASCII(soc, filename);
                HTWriteASCII(soc, " keyword index</TITLE>\n");
                if (*filename) {
                    HTWriteASCII(soc, "This index covers groups: `");
                    HTWriteASCII(soc, filename);
                    HTWriteASCII(soc, "'");
                } else {
                    HTWriteASCII(soc,
"This is the general CERN Computer Center public document index");
                }
                HTWriteASCII(soc,
". Your words will be matched against author-supplied keywords.\n");
                HTWriteASCII(soc,
"<P>Supply keywords to search for information.<P>\n");
                return 0;
            }

            sprintf(buffer, "<IsIndex>\n<TITLE>%s in %s</TITLE>\n",
                keywords, *filename ? filename : "general index" );
            HTWriteASCII(soc, buffer);

            strcpy(system_command,"EXEC NICFIND1 ");
            strcat(system_command, keywords);
            if (*filename) {
                strcat(system_command, " ( ");
                strcat(system_command, filename);
            }
        } else {
            strcpy(system_command,"EXEC NICFOUN1 ");
            strcat(system_command, filename);
        }

    }  /* if FIND */

/*      Formats supported by NEWS are:
**
**      /NEWS                           General news
**      /NEWS/grouplist                 News for given groups
**      /NEWS/grouplist?                All news for all given groups
**      /NEWS/grouplist?keylist         Search news for keywords
**                                      where grouplist = group+group+group.. or voi
**                                      keylist = key+key+key... (can be void)
*/
    else if((
```

```
                (0==strncmp(filename,"NEWS",4))
                || (0==strncmp(filename,"news",4))
            ) && (
                (filename[4]==0)
                || (filename[4]=='/')
                || (filename[4]=='?')
            )) {

        filename=filename+4;
        if (*filename=='/') filename++;          /* Skip first slash */

        {
            char *p;
            while((p=strchr(filename,'+'))!=0) *p=' ';/* Remove +s */
            while((p=strchr(keywords,'+'))!=0) *p=' ';/* Remove +s */

            strcpy(system_command,"EXEC NICFIND1 ");
            if (keywords) strcat(system_command, keywords);
            strcat(system_command, " ( NEWS ");
            if (*filename) {
                strcat(system_command, "(");
                strcat(system_command, filename);
                strcat(system_command, ") ");
            }
        }

        sprintf(buffer, "<IsIndex>\n<TITLE>%s NEWS in %s</TITLE>/n",
            keywords, filename ? filename : "general index" );
        HTWriteASCII(soc, buffer);

/*      Pick up other unknown forms
*/
    } else { /* not FIND or NEWS */
        HTWriteASCII(soc, "Bad syntax. No such document.\n");
        return 0;
    }

/*      Execute system command and get the results
*/
    lines = system(system_command);      /* Number of stacked lines */
    if (TRACE) printf("Command `%s' returned %i lines.\n",
                system_command, lines);
    if (lines<=0) {
        HTWriteASCII(soc,
        "\nSorry, the FIND server could not execute\n  `");
        HTWriteASCII(soc, system_command);
        HTWriteASCII(soc, "'\n\n");
        return 0;
    }

    /*  Copy the file across: */

    for(;lines; lines--){                /* Speed necessary here */
        char *p;
        if (!fgets(buffer, BUFFER_SIZE, stdin))
            sprintf(buffer,
"*** End of file with %d lines left after command\n%s\n - mail BERND@CERNVM",
            lines, system_command);
        for(p=buffer; *p; p++) *p = TOASCII(*p);
        write(soc, buffer, p-buffer);
```

```
    }
    return 0;

} /* HTRetrieve() */
```

```
/*              Handle HTTP request for CERNVM FIND server       FINDGate.c
**
**      This code takes a document name and converts it into
**      a VM CMS command to produce the hypertext the caller requires.
**      This involves, in general, calling an EXEC file which puts
**      the data onto the "stack" The stack allows the data to be read from
**      stdin as though it came from the command device.
**      The commands are written to return a value which is equal to the
**      number of lines put onto the stack.
**
** History
**      29 May 91       Separated from daemon program - TBL
**      19 May 91       Retrieve subroutine format used. -TBL
*/

#define BUFFER_SIZE 4096        /* Arbitrary size for efficiency */

#include "HTUtils.h"
#include "tcp.h"                /* The whole mess of include files */
#include "HTTCP.h"              /* Some utilities for TCP */

extern int WWW_TraceFlag;       /* Control diagnostic output */
extern FILE * LogFile;          /* Log file output */
extern char HTClientHost[16];   /* Client name to be output */
extern int HTWriteASCII(int soc, const char * s);       /* In HTDaemon.c */



/*      Strip white space off a string
**
** On exit,
**      Return value points to first non-white character, or to 0 if none.
**      All trailing white space is overwritten with zero.
*/
PRIVATE char * strip(char * s)
{
#define SPACE(c) ((c==' ')||(c=='\t')||(c=='\n'))
    char * p=s;
    for(p=s;*p;p++);                    /* Find end of string */
    for(p--;p>=s;p--) {
        if(SPACE(*p)) *p=0;     /* Zap trailing blanks */
        else break;
    }
    while(SPACE(*s))s++;        /* Strip leading blanks */
    return s;
}



/*      Handle one message
**      ------------------
**
** On entry,
**      soc             A file descriptor for input and output.
** On exit,
**      returns         >0      Channel is still open.
**                      0       End of file was found, please close file
**                      <0      Error found, please close file.
*/
/*      Retrieve information
```

```
**      --------------------
*/
#ifdef __STDC__
int HTRetrieve(const char * arg, const char * keys, int soc)
#else
int HTRetrieve(arg, keys, soc)
    char *arg;
    char *keys;
    int soc;
#endif
{
#define COMMAND_SIZE    255
#define MAX_LINES 10000                    /* Maximum number of lines returned */
    char command[COMMAND_SIZE+1];
    char keywords[COMMAND_SIZE+1];
    char argument[COMMAND_SIZE+1];
    char buffer[BUFFER_SIZE];
    int status;
    char * filename;     /* Pointer to filename or group list*/
    char system_command[COMMAND_SIZE+1];
    int lines;           /* Number of lines returned by EXEC file */
    int fd;              /* File descriptor number */
    int isIndex = 0;     /* Is the thing asked for an index? */

/*      Remove host and any punctuation. (Could use HTParse @)
*/
    strcpy(argument, arg);
    filename = argument;
    if (argument[0]=='/') {
        if (argument[1]=='/') {
            filename = strchr(argument+2, '/'); /* Skip //host/ */
            if (!filename) filename=argument+strlen(argument);
        } else {
            filename = argument+1;      /* Assume root: skip slash */
        }
    }

    if (!*filename) {
        HTWriteASCII(soc,"<P>Error: No document ID provided.<P>\n");
        return 0;

    }

    if (keys) strcpy(keywords, keys);
    else keywords[0] = 0; /* As if just "?" given (historical) */

/*      Formats supported by FIND are:
**
**      /FIND                           Help for find general index
**      /FIND/                          (same)
**      /FIND/?                         (same)
**      /FIND/group.disk.ft.fn          Get a document from a disk- NOT INDEX
**      /FIND/grouplist?                Help for group index
**      /FIND/grouplist?keylist         Search a set of disks for keywords
**                                      where grouplist = group+group+group.. or voi
**                                      keylist = key+key+key... (not void)
*/
    if((
            (0==strncmp(filename,"FIND",4))
            || (0==strncmp(filename,"find",4))
```

```
        ) && (
            (filename[4]==0)
            || (filename[4]=='/')
            || (filename[4]=='?')
        )) {

        filename=filename+4;
        if (*filename=='/') filename++; /* Skip first slash */

        if (keys || !*filename) {                   /* Index access */
            char *p;
            while((p=strchr(filename,'+'))!=0) *p=' ';/* Remove +s */
            while((p=strchr(keywords,'+'))!=0) *p=' ';/* Remove +s */

            if (!*keywords) {                       /* If no keywords    */
                HTWriteASCII(soc, "<IsIndex><TITLE>");
                HTWriteASCII(soc, filename);
                HTWriteASCII(soc, " keyword index</TITLE>\n");
                if (*filename) {
                    HTWriteASCII(soc, "This index covers groups: `");
                    HTWriteASCII(soc, filename);
                    HTWriteASCII(soc, "'");
                } else {
                    HTWriteASCII(soc,
"This is the general CERN Computer Center public document index");
                }
                HTWriteASCII(soc,
". Your words will be matched against author-supplied keywords.\n");
                HTWriteASCII(soc,
"<P>Supply keywords to search for information.<P>\n");
                return 0;
            }

            sprintf(buffer, "<IsIndex>\n<TITLE>%s in %s</TITLE>\n",
                keywords, *filename ? filename : "general index" );
            HTWriteASCII(soc, buffer);

            sprintf(system_command,"EXEC FSEARCH %s %s",
                HTClientHost, keywords);
            if (*filename) {
                strcat(system_command, " ( ");
                strcat(system_command, filename);
            }
        } else {
            sprintf(system_command,"EXEC FGET %s %s",
                HTClientHost, filename);
        }

    }  /* if FIND */

/*      Formats supported by NEWS are:
**
**      /NEWS                           General news
**      /NEWS/grouplist                 News for given groups
**      /NEWS/grouplist?                All news for all given groups
**      /NEWS/grouplist?keylist         Search news for keywords
**                                      where grouplist = group+group+group.. or voi
**                                      keylist = key+key+key... (can be void)
*/
    else if((
```

```
                (0==strncmp(filename,"NEWS",4))
                || (0==strncmp(filename,"news",4))
            ) && (
                (filename[4]==0)
                || (filename[4]=='/')
                || (filename[4]=='?')
            )) {

        filename=filename+4;
        if (*filename=='/') filename++;          /* Skip first slash */

        {
            char *p;
            while((p=strchr(filename,'+'))!=0) *p=' ';/* Remove +s */
            while((p=strchr(keywords,'+'))!=0) *p=' ';/* Remove +s */

            sprintf(system_command,"EXEC FSEARCH %s", HTClientHost);
            if (keywords) strcat(system_command, keywords);
            strcat(system_command, " ( NEWS ");
            if (*filename) {
                strcat(system_command, "(");
                strcat(system_command, filename);
                strcat(system_command, ") ");
            }
        }

        sprintf(buffer, "<IsIndex>\n<TITLE>%s NEWS in %s</TITLE>/n",
            keywords, filename ? filename : "general index" );
        HTWriteASCII(soc, buffer);

/*      Pick up other unknown forms
*/
    } else { /* not FIND or NEWS */
        HTWriteASCII(soc, "Bad syntax. No such document.\n");
        return 0;
    }

/*      Execute system command and get the results
*/
    lines = system(system_command);      /* Number of stacked lines */
    if (TRACE) printf("Command `%s' returned %i lines.\n",
                system_command, lines);
    if (lines<=0) {
        HTWriteASCII(soc,
        "\nSorry, the FIND server could not execute\n  `");
        HTWriteASCII(soc, system_command);
        HTWriteASCII(soc, "'\n\n");
        return 0;
    }

    /*  Copy the file across: */

    for(;lines; lines--){                /* Speed necessary here */
        char *p;
        if (!fgets(buffer, BUFFER_SIZE, stdin))
            sprintf(buffer,
"*** End of file with %d lines left after command\n%s\n - mail BERND@CERNVM",
            lines, system_command);
        for(p=buffer; *p; p++) *p = TOASCII(*p);
        write(soc, buffer, p-buffer);
```

```
    }
    return 0;

} /* HTRetrieve() */
```

```
/*              Handle HTTP request for CERNVM FIND server      FINDGate.c
**
**      This code takes a document name and converts it into
**      a VM CMS command to produce the hypertext the caller requires.
**      This involves, in general, calling an EXEC file which puts
**      the data onto the "stack" The stack allows the data to be read from
**      stdin as though it came from the command device.
**      The commands are written to return a value which is equal to the
**      number of lines put onto the stack.
**
** History
**      29 May 91       Separated from daemon program - TBL
**      19 May 91       Retrieve subroutine format used. -TBL
*/

#define BUFFER_SIZE 4096        /* Arbitrary size for efficiency */

#include "HTUtils.h"
#include "tcp.h"                /* The whole mess of include files */
#include "HTTCP.h"              /* Some utilities for TCP */

extern int WWW_TraceFlag;       /* Control diagnostic output  */
extern FILE * LogFile;          /* Log file output */
extern char HTClientHost[16];   /* Client name to be output */
extern int HTWriteASCII(int soc, const char * s);       /* In HTDaemon.c */


/*      Strip white space off a string
**
** On exit,
**      Return value points to first non-white character, or to 0 if none.
**      All trailing white space is overwritten with zero.
*/
PRIVATE char * strip(char * s)
{
#define SPACE(c) ((c==' ')||(c=='\t')||(c=='\n'))
    char * p=s;
    for(p=s;*p;p++);                    /* Find end of string */
    for(p--;p>=s;p--) {
        if(SPACE(*p)) *p=0;     /* Zap trailing blanks */
        else break;
    }
    while(SPACE(*s))s++;        /* Strip leading blanks */
    return s;
}


/*      Handle one message
**      ------------------
**
** On entry,
**      soc             A file descriptor for input and output.
** On exit,
**      returns         >0      Channel is still open.
**                      0       End of file was found, please close file
**                      <0      Error found, please close file.
*/
/*      Retrieve information
```

```
**      --------------------
*/
#ifdef __STDC__
int HTRetrieve(const char * arg, const char * keys, int soc)
#else
int HTRetrieve(arg, keys, soc)
    char *arg;
    char *keys;
    int soc;
#endif
{
#define COMMAND_SIZE    255
#define MAX_LINES 10000                     /* Maximum number of  lines returned */
    char command[COMMAND_SIZE+1];
    char keywords[COMMAND_SIZE+1];
    char argument[COMMAND_SIZE+1];
    char buffer[BUFFER_SIZE];
    int status;
    char * filename;    /* Pointer to filename or group list*/
    char system_command[COMMAND_SIZE+1];
    int lines;          /* Number of lines returned by EXEC file */
    int fd;             /* File descriptor number */
    int isIndex = 0;    /* Is the thing asked for an index? */

/*      Remove host and any punctuation. (Could use HTParse @)
*/
    strcpy(argument, arg);
    filename = argument;
    if (argument[0]=='/') {
        if (argument[1]=='/') {
            filename = strchr(argument+2, '/'); /* Skip //host/ */
            if (!filename) filename=argument+strlen(argument);
        } else {
            filename = argument+1;      /* Assume root: skip slash */
        }
    }

    if (!*filename) {
        HTWriteASCII(soc,"<P>Error: No document ID provided.<P>\n");
        return 0;

    }

    if (keys) strcpy(keywords, keys);
    else keywords[0] = 0; /* As if just "?" given (historical) */

/*      Formats supported by FIND are:
**
**      /FIND                               Help for find general index
**      /FIND/                              (same)
**      /FIND/?                             (same)
**      /FIND/group.disk.ft.fn              Get a document from a disk- NOT INDEX
**      /FIND/grouplist?                    Help for group index
**      /FIND/grouplist?keylist             Search a set of disks for keywords
**                                          where grouplist = group+group+group.. or voi
**                                          keylist = key+key+key... (not void)
*/
    if((
                (0==strncmp(filename,"SPIRES",6))
                || (0==strncmp(filename,"spires",6))
```

```
            ) && (
                (filename[6]==0)
                || (filename[6]=='/')
                || (filename[6]=='?')
            )) {

            filename=filename+6;
            if (*filename=='/') filename++; /* Skip first slash */

            if (keys || !*filename) {                   /* Index access */
                char *p;
                while((p=strchr(filename,'+'))!=0) *p=' ';/* Remove +s */
                while((p=strchr(keywords,'+'))!=0) *p=' ';/* Remove +s */

                if (!*keywords) {                       /* If no keywords    */
                    HTWriteASCII(soc, "<IsIndex><TITLE>");
                    HTWriteASCII(soc, filename);
                    HTWriteASCII(soc, " keyword index</TITLE>\n");
                    if (*filename) {
                        HTWriteASCII(soc, "This index covers groups: `");
                        HTWriteASCII(soc, filename);
                        HTWriteASCII(soc, "'");
                    } else {
                        HTWriteASCII(soc,
"This is the general CERN Computer Center public document index");
                    }
                    HTWriteASCII(soc,
". Your words will be matched against author-supplied keywords.\n");
                    HTWriteASCII(soc,
"<P>Supply keywords to search for information.<P>\n");
                    return 0;
                }

                sprintf(buffer, "<IsIndex>\n<TITLE>%s in %s</TITLE>\n",
                    keywords, *filename ? filename : "general index" );
                HTWriteASCII(soc, buffer);

                sprintf(system_command,"EXEC FSEARCH %s %s",
                    HTClientHost, keywords);
                if (*filename) {
                    strcat(system_command, " ( ");
                    strcat(system_command, filename);
                }
            } else {
                sprintf(system_command,"EXEC FGET %s %s",
                    HTClientHost, filename);
            }

        }  /* if FIND */

/*      Formats supported by NEWS are:
**
**      /NEWS                               General news
**      /NEWS/grouplist                     News for given groups
**      /NEWS/grouplist?                    All news for all given groups
**      /NEWS/grouplist?keylist             Search news for keywords
**                                          where grouplist = group+group+group.. or voi
**                                          keylist = key+key+key... (can be void)
*/
        else if((
```

```
            (0==strncmp(filename,"NEWS",4))
            || (0==strncmp(filename,"news",4))
        ) && (
            (filename[4]==0)
            || (filename[4]=='/')
            || (filename[4]=='?')
        )) {

        filename=filename+4;
        if (*filename=='/') filename++;          /* Skip first slash */

        {
            char *p;
            while((p=strchr(filename,'+'))!=0) *p=' ';/* Remove +s */
            while((p=strchr(keywords,'+'))!=0) *p=' ';/* Remove +s */

            sprintf(system_command,"EXEC FSEARCH %s", HTClientHost);
            if (keywords) strcat(system_command, keywords);
            strcat(system_command, " ( NEWS ");
            if (*filename) {
                strcat(system_command, "(");
                strcat(system_command, filename);
                strcat(system_command, ") ");
            }
        }

        sprintf(buffer, "<IsIndex>\n<TITLE>%s NEWS in %s</TITLE>/n",
            keywords, filename ? filename : "general index" );
        HTWriteASCII(soc, buffer);

/*      Pick up other unknown forms
*/
    } else { /* not FIND or NEWS */
        HTWriteASCII(soc, "Bad syntax. No such document.\n");
        return 0;
    }

/*      Execute system command and get the results
*/
    lines = system(system_command);      /* Number of stacked lines */
    if (TRACE) printf("Command `%s' returned %i lines.\n",
                system_command, lines);
    if (lines<=0) {
        HTWriteASCII(soc,
        "\nSorry, the FIND server could not execute\n  `");
        HTWriteASCII(soc, system_command);
        HTWriteASCII(soc, "'\n\n");
        return 0;
    }

    /*  Copy the file across: */

    for(;lines; lines--){                    /* Speed necessary here */
        char *p;
        if (!fgets(buffer, BUFFER_SIZE, stdin))
            sprintf(buffer,
"*** End of file with %d lines left after command\n%s\n - mail BERND@CERNVM",
            lines, system_command);
        for(p=buffer; *p; p++) *p = TOASCII(*p);
        write(soc, buffer, p-buffer);
```

```
    }
    return 0;

} /* HTRetrieve() */
```

/* */ Trace R Arg comm '(' options parse var options subfile . upper subfile Queue '

'

```
Parse var comm . token1 rest
if Abbrev('FIND',token1,3) = 0 Then temp = 'FIND 'token1 rest
else temp = token1 rest
Say temp '('options
if subfile = 'SPIRES' Then subfile = 'HEP'
'EXEC QSPIRES' temp '( STACK NOSTAR OUTPUT TYPE BRIEF IN 'subfile
Exit Queued()
```

/* */ Trace Off Arg comm '(' options parse var options subfile . upper subfile Queue '

```
'
Parse var comm . token1 rest
if Abbrev('FIND',token1,3) = 0 Then temp = 'FIND 'token1 rest
else temp = token1 rest
parse var temp token1 token2 rest
upper token2
Say temp '('options
if subfile = 'SPIRES' Then subfile = 'HEP'
Select
   When token2 = 'REACCESS' Then Do
       Address CMS 'Access 192 B'
       Queue 'Disk re-accessed RC='rc
       End
   When Abbrev('SHOW',token2,3) > 0 Then Do
       'EXEC QSPIRES SHOW 'rest '(STACK NOSTAR IN 'subfile
   End
   When Find('EXPLAIN WHOIS WHATIS WHEREIS QUERY',token2) > 0 Then Do
       'EXEC QSPIRES 'token2 rest '(STACK NOSTAR '
   End
   When Abbrev('BROWSE',token2,3) > 0 Then Do
       'EXEC QSPIRES BROWSE 'rest '(STACK NOSTAR IN 'subfile
       If queued() = 2 Then Do
          parse pull header
          parse pull string
          parse var string first second
             Queue header
             Queue string
          If first = 'Invalid' Then Do
             Queue ' '
             Queue 'Try:  SHOW INDEX for a list of valid terms'
             Queue 'Then: BROWSE term value'
             Queue 'i.e.: BROWSE AUTHOR DRELL'
           End
       End
 End
 When subfile = 'STORES' Then Do
       Parse var temp . temp
       'EXEC STORES' temp '( FIFO NOSTAR'
 End
 Otherwise Do
       'EXEC QSPIRES' temp,
       '( STACK NOSTAR OUTPUT TYPE BRIEF IN 'subfile
       If queued() = 2 Then Do
          parse pull header
          parse pull string
          parse var string first second
             Queue header
             Queue string
          If first = 'Invalid' Then Do
             Queue ' '
             Queue 'Try:  SHOW INDEX for a list of valid terms'
             Queue 'Then: FIND term value'
             Queue 'i.e.: FIND AUTHOR DRELL'
           End
      End
End
```

```
End
Exit Queued()
```

```
/* FIND server version, June 1991
/* trace r */                                                    */
 arg arg
 say "FSEARCH: received ->" arg
 address command; arg host inp; rc=0
 Say '...FSEARCH  Start->'inp'.................'
 parse var inp k_ '(' disks
 if substr(inp,1,1)='!' Then CALL SPECIAL
'GLOBALV SELECT SPECIAL GET TRC'; If trc='I'Then trace i;else trace off
/*_____________________ Initialization ____________________________*/
 null=d2c(0)
 If disks='' Then disks='PUB USER'          /* Later done by client      */
 indsk='Z'                                  /* OK for single index disk  */
 parse var disks d_1 .                      /* To distinguish NEWS/non-N */
 k_=translate(k_,'O','0')                   /* Zeroes and Os alike       */
 k_=translate(k_,' ','.-')                  /* Suppress dots, hyphens    */
 If pos('(',disks) > 0 Then Do              /* Cope with NEWS: start     */
  parse var disks _.1 '('cats.1')'_.2'(' cats.2 ')'_.3'('cats.3'       )'
  ng=0
  Do k=1 to 3; If _.k=''Then leave
    do kk=1 to words(cats.k)
      ng=ng+1;disk.ng=strip(_.k); k_.ng=':'word(cats.k,kk)' 'k_
    end kk
  End
 End /* of '(' inside disks                / Cope with NEWS: end       */
 Else Do /*                                / Normal disk list: start   */
   ng=words(disks)
  'PIPE var disks | split | stem disk.'
   k_.=k_
 End /*                                    / Normal disk list: end     */
 If k_.1='' Then Call OUT 1 'You must give at least one keyword'
 nk=words(k_.1)
/*_____________ Getting all the exp-line numbers ____________________*/
kanz.=0; ktot=0; numeric digits 40
Do ig=1 to ng; g=disk.ig; lm=99999
  If ^FEXIST('FIND$KEY 'g indsk) Then Call OUT 8 "FIND group: '"g"' unknown"
  If QFILE('FIND$KEY 'g indsk,'RECNO')<10000 Then hsh=4999;Else hsh=49999
  Do k=1 to nk
    parse value word(k_.ig,k) with y 13;h=c2d(y)//hsh+3
   'EXECIO 1 DISKR FIND$KEY 'g indsk h' (VAR T'
    parse var t ys''rest; n=find(ys,y)
    If n=0 Then do; lm=0;leave; end
    Else Do
      nn=substr(rest,n*2-1,4);parse var nn n1 3 n2 5
      n1=c2d(n1);n2=c2d(n2)-n1;a.k=substr(rest,n1,n2)
      if lm>n2 Then do; lm=n2; km=k; end
    End
  End k
  Do k=2 to lm by 2;parse var a.km t 3 a.km
   Do i=1 to nk;if i=km Then iterate
     g=pos(t,a.i); If g=0 Then leave
     If g//2^=1 Then do
       s=substr(a.i,g-1)
       if s>t then do;g=0;leave;end
       a.i=substr(a.i,g+1);i=i-1
     End
     Else a.i=substr(a.i,g+2)
   End i;if g=0 & nk>1 Then Iterate
   ktot=ktot+1; kanz.ig=kanz.ig+1;kanz=kanz.ig;b.ig.kanz=c2d(t)
 End k
```

```
  End ig
  If ktot=0 Then Call OUT 4 k_' not found in the FIND groups: 'disks
  numeric digits
  /*._____________ Turning line numbers into exp-lines _____________*/
  nw=0
  Do ig=1 to ng; kanz=kanz.ig; g=disk.ig
    lastl.='';ln.0=null; lastl.0=null
    Do kk=1 to kanz
     'EXECIO 1 DISKR FIND$EXP 'g indsk b.ig.kk '(VAR _'
      If rc>0 Then call x_ rc
      parse var _ _(null)levels
      parse var levels lv 2 ln.1 4 ln.2 6 ln.3 8 ln.4 10; lv=c2d(lv)-1
      If lastl.lv=ln.lv Then Do; nw=nw+1; out.nw=_; end
      Else Do llv=lv-1 to 0 by -1
        If lastl.llv=ln.llv Then Do
         Do lllv=llv+1 to lv
           line=c2d(ln.lllv); lastl.lllv=ln.lllv
          'EXECIO 1 DISKR FIND$EXP 'g indsk line '(VAR __'
           If rc>0 Then call x_ rc
           nw=nw+1;parse var __ out.nw (null) .
         End lllv
         nw=nw+1; out.nw=_
         leave
        End
      End llv
    End kk
  End ig
  CALL OUT 0

  /*-------------------------------------------------------------------*/
   OUT: arg rc text

   If rc=0 Then Do ktrim=1 to nw
     If pos('<A HREF',out.ktrim) > 0 Then Do
       parse var out.ktrim _1 '<A HREF=' _2  '>' _3 _4
       _3=translate(_3,' ','-')
       parse var _3 _31 _32
       If datatype(_3,'W') Then _3=right(_3,2)
       queue _1||'<A HREF=/FIND/'||translate(_2,'+',' ')||'>'
       queue _3'  '_4
     End
     Else queue out.ktrim
   End ktrim
   Else Do; ktot=-rc; If text^='' Then queue text; End

   If d_1='NEWS' Then monitor='XNEWSMON'; Else monitor = 'FINDMON'
  'EXECIO 0 CP (STRING SMSG UMON !LOG! 'monitor right(ktot,4) inp
   Say '   FSEARCH  End  -> rc='right(rc,4)' stacksize='right(queued(),4)
   If pos('@',host) > 0 Then parse var host user '@' h1'.'h2'.'h3'.'h4 .
   Else parse var host h1'.'h2'.'h3'.'h4 .
   host=d2x(h1,2)||d2x(h2,2)||d2x(h3,2)||d2x(h4,2)
   monstring=left(host,9)||substr(date(),1,6)' 'substr(time(),1,5)
   monstring=monstring||right(queued(),5)' 'inp
  'EXECIO 1 DISKW FSEARCH LOGSHEET C (VAR MONSTRING'
   Exit queued()

/*====================================================================*/
SPECIAL:
parse var k_  _1 _2 _3
If _1=!SPECIAL_BERND! Then Do
```

```
 'EXEC SPECIAL '_2 _3; Call OUT 99
End
Return 0
```

/* */ Trace R Arg . comm '(' options Queue '

'

```
'EXEC QSPIRES' comm '( STACK'
'EXEC QSPIRES OUTPUT ( TYPE BRIEF STACK'
Exit Queued()
```

SLACVM SPIRES HEP Preprint Database

Search

Perform search using standard SPIRES terms.

Help

Get help for SPIRES

You can search this index. Type the keyword(s) you want to search for:

SLAC SPIRES HEP Preprint database search

Use standard SPIRES search terms such as...

```
find author Perl, M
find title tau and date 1980
```

HEP (High-Energy Physics) is a joint project of the SLAC and DESY Libraries. As of Jan 1992, it included more than 239,000 bibliographic records dating from 1974 to the present.

Clone copies of HEP run under SPIRES at DESY, KEK, and Yukawa Inst. at Kyoto and are kept up to date by nightly updates via BITNET.

HEP is updated daily with new preprints received in the SLAC library and biweekly with new journal articles and conference proceedings papers indexed at the DESY Library. Input is also received via BITNET from CERN, Fermilab, KEK, Yukawa Inst. at Kyoto U., and SSCL (as of March 1990).

HEP includes all SLAC Library preprint and report holdings from 1974 (all SLAC items from 1962+) as well as all journal articles, conference papers, theses, etc. from the DESY High-Energy Physics Index, a comprehensive bibliography produced at our sister laboratory in Hamburg, Germany.

Here are some examples of typical searches:

-> FIND AUTHOR DORFAN, J.

-> FIND TITLE BLACK HOLE# AND DATE AFTER 1988

SLACVM SPIRES HEP Preprint Database

Search

Perform search using standard SPIRES terms.

Help

Get help for SPIRES

You can search this index. Type the keyword(s) you want to search for:

SLAC SPIRES HEP Preprint database search

Use standard SPIRES search terms such as...

```
find author Perl, M
find title tau and date 1980
```

```
/*              TCP/IP based server for HyperText                    daemon.c
**              ---------------------------------
**
**  History:
**       2 Oct 90       Written TBL. Include filenames for VM from RTB.
**       11 Feb 91      News access added TBL
*/
/* (c) CERN WorldWideWeb project 1990,91. See Copyright.html  for details */

#define VERSION         "v0.2"

/*      Module parameters:
**      ------------------
**
**  These may be undefined and redefined by syspec.h
*/

#define LISTEN_BACKLOG 2          /* Number of pending connect  requests (TCP)*/
#define MAX_CHANNELS 20           /* Number of channels we will  keep open */
#define WILDCARD '*'              /* Wildcard used in addressing */
#define FIRST_TCP_PORT  5000      /* When using dynamic allocation */
#define LAST_TCP_PORT 5999

#define RULE_FILE                 "/etc/httpd.conf"

#include "HTUtils.h"
#include "tcp.h"                  /* The whole mess of include  files */
#include "HTTCP.h"                /* Some utilities for TCP */

#ifdef RULES                      /* Use rules? */
#include "HTRules.h"
#endif

#ifdef __STDC__
extern int HTRetrieve(char * arg, char * keywords, int soc);     /* Handle one request
#else
extern int HTRetrieve();          /* Handle one request */
#endif

PUBLIC char HTClientHost[16];     /* Peer internet address */

/*      Module-Global Variables
**      -----------------------
*/
PRIVATE enum role_enum {master, slave, transient, passive} role;
PRIVATE struct sockaddr_in  soc_address;  /* See netinet/in.h                  */
PRIVATE int                 soc_addrlen;                                      */
PRIVATE int                 master_soc;   /* inet socket number to  listen on  */
PRIVATE int                 com_soc;      /* inet socket number to  read on    */
PRIVATE fd_set              open_sockets; /* Mask of channels which  are active */
PRIVATE int                 num_sockets;  /* Number of sockets to  scan */
PRIVATE BOOLEAN             dynamic_allocation; /* Search for port?  */


/*      Program-Global Variables
**      ------------------------
*/

PUBLIC int    WWW_TraceFlag;    /* Control flag for diagnostics */
PUBLIC FILE * LogFile = 0; /* terryh* Log file if any */
```

```
PUBLIC FILE * logfiles = 0; /* terryh* Log file if any (for WAIS code) */
PUBLIC char * log_file_name = 0;/* Log file name if any (WAIS code)  */


/*      Strip white space off a string
**
** On exit,
**      Return value points to first non-white character, or to 0 if none.
**      All trailing white space is overwritten with zero.
*/
PRIVATE char * strip(char * s)
{
#define SPACE(c) ((c==' ')||(c=='\t')||(c=='\n')||(c=='\r'))     /*  DMX */
    char * p=s;
    for(p=s;*p;p++);                        /* Find end of string */
    for(p--;p>=s;p--) {
        if(SPACE(*p)) *p=0;      /* Zap trailing blanks */
        else break;
    }
    while(SPACE(*s))s++;         /* Strip leading blanks */
    return s;
}


/*      Send a string down a socket
**      ---------------------------
**
**      The trailing zero is not sent.
**      There is a maximum string length.
*/
PUBLIC int HTWriteASCII(int soc, char * s)
{
#ifdef NOT_ASCII
    char * p;
    char ascii[255];
    char *q = ascii;
    for (p=s; *p; p++) {
        *q++ = TOASCII(*p);
    }
    return NETWRITE(soc, ascii, p-s);
#else
    return NETWRITE(soc, s, strlen(s));
#endif
}



/*              Bind to a TCP port
**              ------------------
**
** On entry,
**      tsap    is a string explaining where to take data from.
**              ""      means data is taken from stdin.
**              "*:1729" means "listen to anyone on port 1729"
**
** On exit,
**      returns         Negative value if error.
*/
int do_bind(const char * tsap)
{
```

```
        FD_ZERO(&open_sockets);      /* Clear our record of open sockets */
        num_sockets = 0;


    /*  Deal with PASSIVE socket:
    **
    **      A passive TSAP is one which has been created by the inet daemon.
    **      It is indicated by a void TSAP name.  In this case, the inet
    **      daemon has started this process and given it, as stdin, the connection
    **      which it is to use.
    */
        if (*tsap == 0) {                           /* void tsap => passive */

            dynamic_allocation = FALSE;                 /* not dynamically allocated */
            role = passive; /* Passive: started by daemon */

    #ifdef vms
            {   unsigned short channel;             /* VMS I/O channel */
                struct string_descriptor {          /* This is NOT a proper descriptor*/
                        int size;                   /*  but it will work.
                  */
                        char *ptr;                  /* Should be word,byte,byte,long  */
                } sys_input = {10, "SYS$INPUT:"};
                int status;                 /* Returned status of assign */
                extern int sys$assign();

                status = sys$assign(&sys_input, &channel, 0, 0);
                com_soc = channel;  /* The channel is stdin */
                CTRACE(tfp, "IP: Opened PASSIVE socket %d\n", channel);
                return -BAD(status);
            }
    #else
            com_soc = 0;            /* The channel is stdin */
            CTRACE(tfp, "IP: PASSIVE socket 0 assumed from inet daemon\n");
            return 0;                   /* Good */
    #endif

    /*  Parse the name (if not PASSIVE)
    */
        } else {                                    /* Non-void TSAP */
            char *p;                        /* pointer to string */
            char *q;
            struct hostent  *phost;         /* Pointer to host - See netdb.h */
            char buffer[256];                   /* One we can play with */
            register struct sockaddr_in* sin = &soc_address;

            strcpy(buffer, tsap);
            p = buffer;

    /*  Set up defaults:
    */
            sin->sin_family = AF_INET;              /* Family = internet,  host order  */
            sin->sin_port = 0;                      /* Default: new port,     */
            dynamic_allocation = TRUE;              /*  dynamically allocated  */
            role = passive;                         /*  by default */

    /*  Check for special characters:
    */
            if (*p == WILDCARD) {               /* Any node */
```

```
            role = master;
            p++;
        }

    /*  Strip off trailing port number if any:
    */
        for(q=p; *q; q++)
            if (*q==':') {
                int status;
                *q++ = 0;                   /* Terminate node string */
                sin->sin_port = htons((unsigned short)HTCardinal(
                                        &status, &q, (unsigned int)65535));
                if (status<0) return status;

                if (*q) return -2;  /* Junk follows port number */
                dynamic_allocation = FALSE;
                break;      /* Exit for loop before we skip the zero */
            } /*if*/

    /* Get node name:
    */
        if (*p == 0) {
            sin->sin_addr.s_addr = INADDR_ANY; /* Default: any address */

        } else if (*p>='0' && *p<='9') {   /* Numeric node address:
*/
            sin->sin_addr.s_addr = inet_addr(p); /* See arpa/inet.h */

        } else {                            /* Alphanumeric node name: */
            phost=gethostbyname(p);     /* See netdb.h */
            if (!phost) {
                CTRACE(tfp, "IP: Can't find internet node name `%s'.\n",p);
                return HTInetStatus("gethostbyname");  /* Fail? */
            }
            memcpy(&sin->sin_addr, phost->h_addr, phost->h_length);
        }

        CTRACE(tfp,

            "Daemon: Parsed address as port %d, inet %d.%d.%d.%d\n",
                    (unsigned int)ntohs(sin->sin_port),
                    (int)*((unsigned char *)(&sin->sin_addr)+0),
                    (int)*((unsigned char *)(&sin->sin_addr)+1),
                    (int)*((unsigned char *)(&sin->sin_addr)+2),
                    (int)*((unsigned char *)(&sin->sin_addr)+3));

    } /* scope of p */


/*  Master socket for server:
*/
    if (role == master) {

/*  Create internet socket
*/
        master_soc = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);


        if (master_soc<0)
            return HTInetStatus("socket");
```

```
        CTRACE(tfp, "IP: Opened socket number %d\n", master_soc);

  /*  If the port number was not specified, then we search for  a free one.
  */
        if (dynamic_allocation) {
            unsigned short try;
            for (try=FIRST_TCP_PORT; try<=LAST_TCP_PORT; try++) {

                soc_address.sin_port = htons(try);
                if (bind(master_soc,
                        (struct sockaddr*)&soc_address,
                                /* Cast to generic sockaddr */
                        sizeof(soc_address)) == 0)
                    break;
                if (try == LAST_TCP_PORT)
                    return HTInetStatus("bind");
            }
            CTRACE(tfp, "IP:  Bound to port %u.\n",
                    ntohs(soc_address.sin_port));
        } else {                                        /* Port was specified */
            if (bind(master_soc,
                     (struct sockaddr*)&soc_address,    /* Case to generic address */
                     sizeof(soc_address))<0)
                return HTInetStatus("bind");
        }
        if (listen(master_soc, LISTEN_BACKLOG)<0)
            return HTInetStatus("listen");

        CTRACE(tfp, "Daemon: Master socket(), bind() and listen() all OK\n");
        FD_SET(master_soc, &open_sockets);
        if ((master_soc+1) > num_sockets) num_sockets=master_soc+1;

        return master_soc;
    } /* if master */


    return -1;          /* unimplemented role */

} /* do_bind */


/*      Handle one message
**      ------------------
**
** On entry,
**      soc             A file descriptor for input and output.
** On exit,
**      returns         >0      Channel is still open.
**                      0       End of file was found, please close file
**                      <0      Error found, please close file.
*/
PUBLIC int HTHandle(int soc)
{
#define COMMAND_SIZE    255
#define MAX_LINES 10000                 /* Maximum number of lines returned */
    char command[COMMAND_SIZE+1];
    char *keywords;     /* pointer to keywords in address */
```

```
    int status;
    char * arg;          /* Pointer to the argument string */

    time_t theTime;      /* A long integer giving the datetime in secs */

    for (;;) {
        status = read(soc, command, COMMAND_SIZE);
        if (status<=0) {
            if (TRACE) printf("read returned %i, errno=%i\n", status, errno);
            return status;       /* Return and close file */
        }
        command[status]=0;       /* terminate the string */
#ifdef NOT_ASCII
        {
            char * p;
            for (p=command; *p; p++) {
                *p = FROMASCII(*p);
                if (*p == '\n') *p = ' ';
            }
        }
#endif

/*      Log the call:
*/
        if (LogFile) {
            time(&theTime);
            fprintf(LogFile, "%24.24s %s %s",
                ctime(&theTime), HTClientHost, command);
            fflush(LogFile);    /* Actually update it on disk */
            if (TRACE) printf("Log: %24.24s %s %s",
                ctime(&theTime), HTClientHost, command);
        }

        arg=strchr(command,' ');
        if (arg) {
            *arg++ = 0;                                 /* Terminate command
        */
            arg = strip(arg);                           /* Strip space
        */

            if (0==strcmp("GET", command)) {    /* Get a file
        */
                keywords=strchr(arg, '?');
                if (keywords) *keywords++=0;    /* Split into two */
                return HTRetrieve(arg, keywords, soc);
            }

        } /* Space found supplied */

        HTWriteASCII(soc, "<h1>HTTP Server v1.02</h1><h2>Test response</h2>\n");
        HTWriteASCII(soc, "\n\nThe (unrecognised) command used was `");
        HTWriteASCII(soc, command);
        HTWriteASCII(soc, "'.\n\n");

        if (TRACE) printf("Return test string written back'\n");
        return 0;                   /* End of file - please close socket */
    } /* for */
```

```
    } /* handle */


    /*      Handle incomming messages
    server_loop()
    **      ------------------------
    **
    ** On entry:
    **
    **      timeout         -1 for infinite, 0 for poll, else in  units of 10ms
    **
    ** On exit,
    **      returns         The status of the operation, <0 if failed.
    **
    */
    #ifdef __STDC__
    PRIVATE int server_loop(void)
    #else
    PRIVATE int server_loop()
    #endif
    {
        int tcp_status;             /* <0 if error, in general */
        int timeout = -1;           /* No timeout required but code exists */
        for(;;) {

    /*  If it's a master socket, then find a slave:
    */
            if (role == master) {
    #ifdef SELECT
                fd_set              read_chans;
                fd_set              write_chans;
                fd_set              except_chans;
                int                 nfound;     /* Number of ready channels */
                struct timeval      max_wait;   /* timeout in form for select() */


                FD_ZERO(&write_chans);          /* Clear the write mask */
                FD_ZERO(&except_chans);         /* Clear the exception mask */
    /*  If timeout is required, the timeout structure is set up. Otherwise
    **  (timeout<0) a zero is passed instead of a pointer to the struct timeval.
    */
                if (timeout>=0) {
                    max_wait.tv_sec = timeout/100;
                    max_wait.tv_usec = (timeout%100)*10000;
                }


                for (com_soc=-1; com_soc<0;) {      /* Loop while connections keep coming


    /*  The read mask expresses interest in the master channel for incomming
    **  connections) or any slave channel (for incomming messages).
    */

    /*  Wait for incoming connection or message
    */
```

```
                read_chans = open_sockets;      /* Read on all active channels */
                if (TRACE) printf(
    "Daemon: Waiting for connection or message. (Mask=%x hex, max=%x hex).\n",

                        *(int *)(&read_chans), num_sockets);
                nfound=select(num_sockets, &read_chans,
                    &write_chans, &except_chans,
                    timeout >= 0 ? &max_wait : 0);

                if (nfound<0) return HTInetStatus("select()");
                if (nfound==0) return 0;        /* Timeout */

    /*      We give priority to existing connected customers. When there are
    **      no outstanding commands from them, we look for new customers.
    */
    /*      If a message has arrived on one of the channels, take that channel:
    */
                {
                    int i;
                    for(i=0; i<num_sockets; i++)
                        if (i != master_soc)
                            if (FD_ISSET(i, &read_chans)) {
                            if (TRACE) printf(
                                "Message waiting on socket %i\n", i);
                            com_soc = i;                /* Got one! */
                            break;
                        }
                    if (com_soc>=0) break; /* Found input socket */


                } /* block */

    /*  If an incomming connection has arrived, accept the new socket:
    */
                if (FD_ISSET(master_soc, &read_chans)) {


                        CTRACE(tfp, "Daemon: New incomming connection:\n");
                        tcp_status = accept(master_soc,
                                        (struct sockaddr *)&soc_address,
                                        &soc_addrlen);
                        if (tcp_status<0)
                            return HTInetStatus("accept");
                        CTRACE(tfp, "Daemon: Accepted new socket %d\n",
                            tcp_status);
                        FD_SET(tcp_status, &open_sockets);
                        if ((tcp_status+1) > num_sockets)
                                num_sockets=tcp_status+1;


                } /* end if new connection */



            } /* loop on event */

#else   /* SELECT not supported */
```

```
                if (com_soc<0) { /* No slaves: must accept */
                        CTRACE(tfp,

                        "Daemon: Waiting for incomming connection...\n");
                        tcp_status = accept(master_soc,
                                        &rsoc->mdp.soc_tcp.soc_address,
                                        &rsoc->mdp.soc_tcp.soc_addrlen);
                        if (tcp_status<0)
                            return HTInetStatus("accept");
                        com_soc = tcp_status;       /* socket number */
                        CTRACE(tfp, "Daemon: Accepted socket %d\n", tcp_status);
                } /* end if no slaves */


    #endif

            }  /* end if master */


    /* com_soc is now valid for read */

            {
                struct sockaddr_in addr;
                int namelen = sizeof(addr);
                getpeername(com_soc, (struct sockaddr*)&addr, &namelen);
                strncpy(HTClientHost,
                    inet_ntoa(addr.sin_addr), sizeof(HTClientHost));
            }

    /*  Read the message now on whatever channel there is:
    */
            CTRACE(tfp,"Daemon: Reading socket %d from host %s\n",
                    com_soc, HTClientHost);

            tcp_status=HTHandle(com_soc);

            if(tcp_status<=0) {
                if (tcp_status<0) {                         /* EOF or error */
                    CTRACE(tfp,                             /* error */
                    "Daemon: Error %i handling incomming message (errno=%i).\n",
                            tcp_status, errno);
                    /* DONT return HTInetStatus("netread");  error */
                } else {
                    CTRACE(tfp, "Daemon: Socket %d disconnected by peer\n",
                        com_soc);
                }
                if (role==master) {
                    NETCLOSE(com_soc);
                    FD_CLR(com_soc, &open_sockets);
                } else {  /* Not multiclient mode */
    #ifdef VM
                    return -69;
    #else
                    return -ECONNRESET;
    #endif
                }
            } /* end if handler got EOF or error */


        }; /* for loop */
```

```
    /*NOTREACHED*/
    } /* end server_loop */


    /*              Main program
    **              ------------
    **
    **      Options:
    **      -v              verify: turn trace output on to stdout
    **      -a addr         Use different tcp port number and style
    **      -l file         Log requests in ths file
    **      -r file         Take rules from this file
    **      -R file         Clear rules and take rules from file.
    */
    int main(int argc, char*argv[])
    {
        int status;
    #ifdef RULES
        int rulefiles = 0;          /* Count number loaded */
    #endif
        char * addr = "";           /* default address */
        WWW_TraceFlag = 0;                  /* diagnostics off by default */
    #ifdef RULES
        HTClearRules();
    #endif
        {
            int a;


            for (a=1; a<argc; a++) {


                if (0==strcmp(argv[a], "-v")) {
                    WWW_TraceFlag = 1;

                } else if (0==strcmp(argv[a], "-a")) {
                    if (++a<argc) addr = argv[a];
    #ifdef RULES
                } else if (0==strcmp(argv[a], "-r")) {
                    if (++a<argc) {

                        if (HTLoadRules(argv[a]) < 0) exit(-1);
                        rulefiles++;
                    }
                } else if (0==strcmp(argv[a], "-R")) {
                    rulefiles++;                /* Inhibit rule file load */
    #endif
                } else if (0==strcmp(argv[a], "-l")) {
                    if (++a<argc) {
                        LogFile = fopen(argv[a], "a");
                        logfiles = LogFile;             /* name for WAIS */
                        log_file_name = argv[a];        /* For WAIS code */
                    }
                    if (!LogFile) {
                        fprintf(stderr,
                            "Can't open log file %s\n", argv[a]);
                        logfiles = LogFile = stderr;
                    }
```

```
    /* Remove because an inet daemon process gets started every request.
                    fprintf(LogFile, "\nhttpd: HTTP Daemon %s started.\n",
                                VERSION);
    */
                } /*ifs */
            } /* for */
        } /* scope of a */


    #ifdef RULES
        if (rulefiles==0) {          /* No mention */
            if (HTLoadRules(RULE_FILE) < 0) exit(-1);
        }
    #endif



        status = do_bind(addr);
        if (status<0) {
            fprintf(stderr, "Daemon: Bad setup: Can't bind and listen on port.\n");
            fprintf(stderr, "     (Possibly server already running, for example).\n");
            exit(status);
        }


        status = server_loop();

        if (status<0) {
            /* printf("Error in server loop.\n");  not error if inetd-started*/
            exit(status);
        }


        exit(0);
        return 0;               /* For gcc */
}
```

```
/*                          Generic Communication Code                  HTTCP.c
**                          ==========================
**
**      This code is in common between client and server sides.
*/


#include "HTUtils.h"
#include "tcp.h"                        /* Defines SHORT_NAMES if necessary */
#ifdef SHORT_NAMES
#define HTInetStatus                    HTInStat
#define HTInetString                    HTInStri
#endif


/*      Module-Wide variables
*/

PRIVATE char *hostname=0;                       /* The name of this host */

/*      PUBLIC VARIABLES
*/

PUBLIC struct sockaddr_in HTHostAddress;        /* The internet address of the host *
                                                /* Valid after call to HTHostName() */
/*      Encode INET status (as in sys/errno.h)                          inet_status()
**      ------------------
**
** On entry,
**      where           gives a description of what caused the error
**      global errno    gives the error number in the unix way.
**
** On return,
**      returns         a negative status in the unix way.
*/
#ifdef vms
extern int uerrno;      /* Deposit of error info (as perr errno.h) */
extern int vmserrno;    /* Deposit of VMS error info */
extern volatile noshare int errno;  /* noshare to avoid PSECT conflict */
#else
#ifndef errno
extern int errno;
#endif
#endif

#ifndef VM
#ifndef vms
#ifndef NeXT
extern char *sys_errlist[];             /* see man perror on cernvax */
extern int sys_nerr;
#endif
#endif
#endif


/*      Report Internet Error
**      ---------------------
*/
```