

hp e3000

strategy



MPE CI Programming for 7.0

... and other tidbits



presented by
Jeff Vance, HP-CSY
jeff_vance@hp.com

hp e3000

outline

(read the notes too!)

strategy

- redo - old features and new
- common CI programming commands
- error handling techniques
- expressions
- evaluator functions -- general
- JINFO, JOBCNT and PINFO functions
- CI variables
- alternate entry points in scripts and UDCs
- I/O redirection techniques
- lots of examples
- appendix

hp e3000

strategy

redo

- delete a word
 - dw, >dw, dwddw, dwiXYZ
- delete up to a special character
 - d., d/, d*, d/iXYZ, d.d
- delete to end-of-line
 - d>
- delete two or more non-adjacent characters
 - d d
- upshift/downshift a character or word
 - ^, ^w, v, vw, >^, >v, ^>, v>
- append to end-of-line
 - >XYZ
- replace starting at end of line
 - >rXYZ
- change one string to another
 - c/ABCD/XYZ, c:123::
- undo last or all edits
 - u, u twice
- available in CI, VOLUTIL, STAGEMAN, DEBUG others...

hp e3000

common CI “programming” commands

strategy

- IF, ELSEIF, ELSE, ENDIF
ESCAPE, RETURN
- WHILE, ENDWHILE looping
- ECHO, INPUT
- SETVAR, DELETEVAR
SHOWVAR
- ERRCLEAR
- RUN
XEQ
- PAUSE
- OPTION recursion

branching

terminal, console, file I/O

create/modify/delete/display a variable

sets CI error variables to 0

invoke a program
invoke a program or script

sleep; job synchronization

only way to get recursion in UDCs

error handling

- use **HPAUTOCONT** variable judiciously
 - better --
continue
command
if hpcierr <> 0 then ...
- if error-condition then
echo something...
return -- or -- escape
endif
...
- RETURN vs. ESCAPE
 - return goes back ONE level
 - escape goes back to the CI level in a session, to an active CONTINUE, or can abort a job
- HPCIERRMSG - contains the error text for the value of CIERROR
- :ERRCLEAR - sets HPCIERR, CIERROR, HPFSERR, HPCIERRCOL to zero

CI expressions

- what is an expression?
 - any variable, constant or function with or without an operator, e.g:
MYVAR, "a"+"b", $x^{10} \cdot y / (j \bmod 6)$, false, $(x > \text{lim})$ or $(\text{input()} = "y")$
 - partial evaluation:
 - if **true** or x # "x" side not evaluated
 - if **false** and x # "x" side not evaluated
 - if bound(z) and $z > 10$ then # if "z" not defined it won't be referenced
 - problems when MPEX runs the script
- where can expressions be used?
 - 5 commands that accept implicit variable references:
:calc, :if, :elseif, :setvar, :while
 - `![ expression ]` can be used in any command:
 - :build afile; rec=-80; disc= ![100+varX]
 - :build bfile; disc= ![finfo("afile","eof")*3] # file b is 3 times bigger
- examples:
 - :print ![input("File name? ")]
 - :setvar reply ups(rtrim(ltrim(reply)))

CI functions

- functions are invoked by their name, accept zero or more parms and return a value in place of their name and arguments
- file oriented functions:
 - BASENAME, DIRNAME, FINFO, FSYNTAX, FQUALIFY
- string parsing functions:
 - ALPHA, ALPHANUM, DELIMPOS, DWNS, EDIT, LEN, LFT, LTRIM, NUMERIC, PMATCH, POS, REPL, RHT, RPT, RTRIM, STR, UPS, WORD, WORDCNT, XWORD
- conversion functions:
 - CHR, DECIMAL, HEX, OCTAL, ORD
- arithmetic functions
 - ABS, MAX, MIN, MOD, ODD
- job/process functions:
 - JINFO, JOBCNT, PINFO
- misc. functions:
 - ANYPARM, BOUND, INPUT, SETVAR, TYPEOF

JINFO function

syntax: JINFO (“[#]S|Jnnnn”, “item” [,status])

where jobID can be “[#]J|Snnn” or “0”, meaning “me”

- 63 unique items: Exists, CPUsec, IPAddr, JobQ, Command, JobUserAcctGroup, JobState, StreamedBy, Waiting ...
- status parm is a variable name. If passed, CI sets status to JINFO error return -- normal CI error handling bypassed
- can see non-sensitive data for any job on system
- can see sensitive data on: “you”; on other jobs w/ same user.acct if jobsecurity is LOW; on other jobs in same acct if AM cap; on any job if SM or OP cap

JOBCNT function

syntax: JOBCNT ("job_spec" [,joblist_var])

- "Job_Spec" can be:
 - "user.account"
 - "jobname,user.account"
 - "@J", "@S", "@"
 - "@J:[jobname,]user.acct" or "@S:[jobname,]user.acct"
 - wildcarding is supported
 - use empty jobname ("") to select jobs without jobnames
 - omit jobname to match any jobname

PINFO function

syntax: PINFO (pin, “item” [,status])

where PIN can be a string, “[#P]nnn[.tin]”, or a simple integer, “0” is “me”

- 66 unique items: Alive, IPAddr, Parent, Child, Children, Proctype, WorkGroup, SecondaryThreads, NumOpenFiles, ProgramName, etc.
- status parm is a variable name. If passed, CI sets status to PINFO error return -- normal CI error handling bypassed
- can see non-sensitive data for any user process on system
- follows SHOWPROC's rules for sensitive data

variable scoping

- all CI variables are job/session global, except the following:
HPAUTOCONT, HPCMDTRACE, HPERRDUMP, HPERRSTOLIST, HPMSGFENCE
- easy to set “persistent” variables via logon UDC
- need care in name of UDC and script “local” variables to not collide with existing job/session variables
 - `_scriptName_varname` -- for all script variable names. Use: `deletevar _scriptName_@` at end of script
 - Can create unique variable names by using `!HPPIN`, `!HPCIDDEPTH`, `!HPUSERCMDEPTH` as part of the name, e.g.
`:setvar _script_xyz_!hppin , value`
- save original value of some “environment” variables
 - `:setvar _script_savemsgfence hpmsgfence`
`:setvar hpmsgfence 2`

variable referencing

- two ways to reference a variable:
 - explicit -- !varName
 - implicit -- varName
- some CI commands expect variables as their arguments, e.g.
 - :CALC, :IF, :ELSEIF, :SETVAR, :WHILE
 - use implicit referencing here, e.g.
:if (HPUSER = "MANAGER") then
- most CI commands don't expect variable names (e.g. BUILD, ECHO, LISTF)
 - use explicit referencing here, e.g.
:echo You are logged on as: !HPUSER.!HPACCOUNT
 - note: all UDC/script parameters must be explicitly referenced
- all CI functions accept variable names, thus implicit referencing works
 - :while JINFO (HPLASTJOB, "exists") do... better than ...
:while JINFO ("!HPLASTJOB", "exists") do

!varname

strategy

- processed by the CI early, before command name is known
 - can cause hard-to-detect bugs in scripts - array example
- loose variable type -- strings need to be quoted, e.g..


```
"!varName"
```
- !! (two exclamation marks) used to "escape" the meaning of "!", multiple "!'s" are folded 2 into 1
 - even number of "!" --> don't reference variable's value
 - odd number of "!" --> reference the variable's value
- useful to convert an ASCII number to an integer, e.g.

setvar int "123"	or	input foo, "enter a number"
if !int > 0 then ...		if !foo = 321 then ...
- the only way to reference UDC or script parameters
- the only way for most CI commands to reference variables

implicit referencing - just varname

- evaluated during the execution of the command -- later than explicit referencing
- makes for more readable scripts
- variable type is preserved -- no need for quotes, like: “!varname”
- only 5 commands accept implicit referencing: CALC, ELSEIF, IF, SETVAR, WHILE -- all others require explicit referencing
- all CI function parameters accept implicit referencing
- variables inside ![expression] may be implicitly referenced
- performance differences:
 - “!HPUSER.!HPACCOUNT” = “OP.SYS” 4340 msec
 - HPUSER + “.” + HPACCOUNT = “OP.SYS” 4370 msec
 - HPUSER = “OP” and HPACCOUNT = “SYS” 4455 msec*
(*with user match true)

my preference is the last choice since many times :IF will not need to evaluate the expression after the AND

entry points

- simple convention for executing same UDC/script starting in different “sections” (or subroutines)
- a UDC/script invokes itself recursively passing in the name of an entry (subroutine) to execute
- the script detects that it should execute an alternate entry and skips all the code not relevant to that entry.
- most useful when combined with I/O redirection, but can provide the appearance of generic subroutines
- benefits are: fewer script files to maintain, slight performance gain since MPE opens an already opened file faster, can use variables already defined in script
- UDCs need `OPTION RECURSION` to use multiple entry points

entry points (cont)

strategy

- two approaches for alternate entries:
 - define a parm to be the entry point name, defaulting to the main part of the code ("main")
 - the UDC/script invokes itself recursively in the main code, and may use I/O redirection here too
 - each entry point returns when done (via :RETURN command)

----- or -----

- test HPSTDIN or HPINTERACTIVE variable to detect if script/UDC has I/O redirected.
- if TRUE then assume UDC/script invoked itself.
- limited only to entry points used when \$STDLIST or \$STDIN are redirected
- limited to a single alternate entry point, may not work well in jobs

entry points (cont)

- generic approach:

```

    PARM p1 ... entry=main
    if "!entry" = "main" then
        ... initialize etc...
        xeq !HPFILE !p1, ... entry=go
        ... cleanup etc...
        return
    elseif "!entry" = "go" then...
        # execute the GO subroutine ...
        return
    elseif "!entry" = ...
        ...
    endif

```

default entry is "main"

run same script, different entry

entry points (cont)

strategy

- i/o redirection specific approach:

```
PARM p1 ...          # no "entry" parm defined
if HPSTDIN = "$STDIN" then
    ... ("main" entry -- initialize etc...)
    xeq !HPFILE !p1, ... <somefile
    ... (cleanup etc...)
    return
else                # no elseif since only 1 alternate
    # execute the entry to read "somefile"
    setvar eof FINFO(hpstdin, "eof")
    ...
    return
endif
```

file i/o

strategy

- three main alternatives:
 - write to (create) and read from a MSG file via I/O redirection
 - use :PRINT and I/O redirection to read file 1 record at a time
 - use entry points and I/O redirection
- why not use INPUT in WHILE to read a flat file, e.g.:
 - while not eof do
 input varname < filename
 ...
endwhile
- MSG file works because each read is destructive, so next INPUT reads next record

file i/o - MSG file

- PARM fileset=./@
 # This script reads a file produced by LISTFILE,6 and measures CPU millisecs
 # using a MSG file
setvar savecpu hpcpumsecs
 errclear
 file msg=/tmp/LISTFILE.msg; **MSG**
 continue
 listfile !fileset,6 >*msg
 if hpcierr = 0 then
 # read listfile names into a variable
 setvar cntr setvar(eof, finfo('*msg', "eof"))
while setvar(cntr, cntr-1) >= 0 do
 input rec <*msg
endwhile
 endif
echo ![hpcpumsecs - savecpu] msecs to read !eof records.
 deletevar cntr, eof, rec

```
:readmsg
259 msec
```

s to read 22 records


```
:readmsg @.pub.sys
15,845 msec
```

s to read 1,515

file i/o - :print

- PARM fileset=./@
 # This script reads a file produced by LISTFILE,6 and measures CPU msec
 # using PRINT as an intermediate step
 setvar savecpu hpcpumsecs
 errclear
 continue
 listfile !fileset,6 > lftemp
 if hpcierr = 0 then
 # read listfile names into a variable
 setvar cntr 0
 setvar eof finfo('lftemp',"eof")
 while setvar(cntr, cntr+1) <= eof do
 print lftemp;start=!cntr;end=!cntr > lftemp1
 input rec <lftemp1
 endwhile
 endif
 echo ![hpcpumsecs - savecpu] msec to read !eof records.
 deletevar cntr,eof,rec

:readprnt

735 msec to read 22 records

3 times slower than MSG files

:readprnt @.pub.sys

74,478 msec to read 1515 recs

over 4 times slower than MSG files!

file i/o - entry points

- PARM fileset=./@, **entry="main"**
This script reads a file produced by LISTFILE,6 and measures CPU msecs
using entry points and script redirection
if "!entry" = "main" then
 setvar savecpu hpcpumsecs
 errclear
 continue
 listfile !fileset,6 > lftemp
 if hpcierr = 0 then
 xeq !hpfile !fileset entry=read <lftemp
 endif
 echo ![hpcpumsecs - savecpu] msecs to read !eof records.
 deletevar cntr,eof,rec
 purge lftemp,temp
 return
 . . . (continued on next slide)

file i/o - entry points (cont)

else

```
# read listfile names into a variable
setvar cntr setvar(eof, finfo(hpstdin, "eof"))
while setvar(cntr,cntr-1) >= 0 and setvar(rec, input()) <> chr(1) do
endwhile
return
endif
```

:readntry

90 msec to read 24 records.

---> Almost 3 times faster than MSG files

---> 8 times faster than the PRINT method!

:readntry @.pub.sys

2400 msec to read 1,515 records.

---> Over 6 times faster than MSG files

---> 31 times faster than using PRINT!

hp e3000

examples

strategy

- a few simple examples
- what version of MPE will run this script?
- easy way to print \$STDLIST spoolfile for a job
- flexible way to change directories (CWD)
- job synchronization example
- INFO= string examples
- create a “random” name or value
- tying many concepts together with the WHERE script
- STREAM UDC - abbreviated

simple examples

strategy

- turn on/off UDC and script command tracing:
 - setvar HPCMDTRACE not(hpcmdtrace) # toggle

- display last N records of a file (no process creation)
 - PARM file, last=12 "Tail" script
print !file; start= -!last

- display CI error text for a CI error number
 - PARM cierr= !cierror "Cierr" script
setvar save_err cierror
setvar cierror !cierr
showvar HPCIERRMSG
setvar cierror save_err
deletevar save_err

more examples

strategy

- alter priority of job just streamed:

```
PARM job=!HPLASTJOB; pri=CS
altproc job=!job; pri=!pri
```

“Altproc” script

-
- Aligned fields for output:

```
PARM cnt=5
setvar i 0
while setvar(i,i+1) <= !cnt do
  setvar a rpt('a',i)
  setvar b rpt('b',!cnt-i+1)
  echo xx ![rpt(' ',!cnt-len(a))]!a xx ![rpt(' ',!cnt-len(b))]!b xx
endwhile
```

“Align” script

- Example:

```
:align 4
xx    a xx  bbbb xx
xx   aa xx   bbb xx
xx  aaa xx    bb xx
xx aaaa xx     b xx
```

hp e3000

MPE version

strategy

- PARM vers_parm=!hprelversion
react to MPE version string
setvar vers "!vers_parm"
convert to integer, e.g.. "C.65.02" => 6502
setvar vers str(vers,3,2) + rht(vers,2)
setvar vers !vers
if vers >= 7000 then
 echo On 7.0!
elseif vers >= 6500 then
 echo On 6.5!
elseif vers >= 6000 then
 echo On 6.0!
endif

“Vers” script

printing spoolfiles

- PRINTSP script:

```
PARM job=!HPLASTJOB
# Prints spoolfile for a job, default is the last job you streamed
if "!job" = "" then
    echo No job to print
    return
endif
setvar hplastjob "!job"
if HPLASTSPID = "" then
    echo No $STDLIST spoolfile to print
    return
endif
print !HPLASTSPID.out.hpspool
```
- `:stream scopejob`
#J324
`:printsp`
:JOB SCOPEJOB,MANAGER.SYS,SCOPE.
Priority = DS; Inpri = 8; Time = UNLIMITED seconds . . .

new location (group, CWD)

- CD script

```

    PARM dir=""
    setvar d "!dir"
    # "-" means go to prior CWD
    if d = '-' and bound(save_chdir) then
        setvar d save_chdir
    elseif fsyntax(d) = "MPE" then
        # MPE syntax?
        # HFS dir?
        if finfo("./"+d, "exists") then
            setvar d "./" + d
        elseif finfo("../"+ups(d), "exists") then
            # MPE group?
            setvar d "../" + ups(d)
        elseif finfo(ups(d), "exists") then
            # MPE dir name?
            setvar d ups(d)
        endif
    endif
    setvar save_chdir HPCWD
    chdir !d
  
```

hp e3000

synchronize jobs

strategy

```
!JOB job0...
!limit +2
!stream job1
!pause job=!hplastjob
!stream job2
!errclear
!pause 600, !hplastjob
!if hpcierr = -9032 then
!   tellop Job "!hplastjob" has exceeded the 10 minute limit
!   eoj
!endif
!stream job3
!pause job=!hplastjob; WAIT
!input reply, "Reply 'Y' for !hplastjob"; readcnt=1; CONSOLE
!if dwns(reply) = "y" then
. . .
```

INFO= example

- ANYPARM info=![""] # "anyrun" script
run volutil.pub.sys; info=":!info"
 - :anyrun echo "Hi there!"
run volutil.pub.sys;info=":echo "Hi there!""

^

Expected semicolon or carriage return. (CIERR 687)
- ANYPARM info=![""]
setvar _inf repl('!info', '', '""') # double up quotes in :RUN
run volutil.pub.sys;info=":_inf "
 - :anyrun echo "Hi there!"
Volume Utility A.02.00, (C) Hewlett-Packard Co.,
1987. All Rights...
volutil: :echo "Hi there!"
"Hi there!"
- is this correct now?

INFO= example (cont)

- ANYPARM info=![""]
setvar _inf anyparm(!info) # note info parm is **not** quoted
setvar _inf repl(_inf, '', '""')
run volutil.pub.sys;info=":_!inf "
- :anyrun echo "Hi there, 'buddy'!"
Volume Utility A.02.00, (C) Hewlett-Packard Co., 1987.
All Rights...
volutil: :echo "Hi there, 'buddy'!"
"Hi there, 'buddy'!"

hp e3000

strategy

random names

- PARM varname, minlen=4, maxlen=8
This script returns in the variable specified as "varname" a `random'
name consisting of letters and numbers - cannot start with a number.
At least "minlen" characters long and not more than "maxlen" chars.

expression for a `random' letter:

```
setvar letter "chr( (hpcpumsecs mod 26) + ord('A') )"
```

expression for a `random' number:

```
setvar number "chr((hpcpumsecs mod 10) + ord('0'))"
```

first character must be a letter

```
setvar !varname !letter
```

now fill in the rest, must have at least "minlen" chars , up to "maxlen"

```
setvar i 1
```

```
setvar limit min( (hpcpumsecs mod !maxlen) + !minlen, !maxlen)
```

```
while setvar(i,i+1) <= limit do
```

```
  if odd(hpcpumsecs) then
```

```
    setvar !varname !varname + !letter
```

```
  else
```

```
    setvar !varname !varname + !number
```

```
  endif
```

```
endwhile
```

where is that “cmd”?

```
PARM cmd="", entry=main
# This script finds all occurrences of "cmd" as a UDC, script or program in
# HPPATH. Wildcards are supported for UDC, program and command file names.
# Note: a cmd name like "foo.sh" is treated as a POSIX name, not a qualified
#       MPE name.
if "!entry" = "main" then
  errclear
  setvar _wh_cmd "!cmd"
  if delimpos(_wh_cmd,"/.") = 1 then
    echo WHERE requires the POSIX cmd to be unqualified.
    return
  endif

  # see if the command could be a UDC (wildcards are supported)
  setvar _wh_udc_ok (delimpos(_wh_cmd,'.') = 0)
  # see if the command could be an MPE filename (wildcards ok, and
  # MPE names cannot be qualified at all)
  setvar _wh_mpe_ok (delimpos(_wh_cmd,'_') = 0)
  ## All command values are assumed to be ok as a POSIX filename.
  ## The dash (-) char is excluded above since it could be in a [a-z] pattern
  . . . continued . . .
```

hp e3000

strategy

where (cont)

```
. . .
# check for UDCs first
if _wh_udc_ok then
    continue
    showcatalog >whereudc
    if cierror = 0 then
        xeq !hpfile !_wh_cmd entry=process_udcs <whereudc
    endif
endif

# Now check for command/program files
if word(setvar(_wh_syn,fsyntax("."+_wh_cmd))) = "ERROR" then
    # illegal name, could be a longer UDC name, in any event there
    # no need to check for command/program files.
    deletevar _wh_@
    return
endif
setvar _wh_wild pos("WILD",_wh_syn) > 0
. . . continued . . .
```

where (cont)

```

...
# loop through hppath
setvar _wh_i 0
while setvar(_wh_tok,word(hppath,";;",setvar(_wh_i,_wh_i+1)))<>" do
  if delimpos(_wh_tok,"/.") = 1 then
    # we have a POSIX path element
    setvar _wh_tok "!_wh_tok/!_wh_cmd"
  elseif _wh_mpe_ok then
    # we have an MPE syntax HPPATH element with an unqualified _tok
    setvar _wh_tok "!_wh_cmd.!_wh_tok"
  endif
  errclear
  if _wh_wild then
    continue
  endif
  listfile !_wh_tok,6 >prntlf
elseif finfo(_wh_tok,'exists') then
  # write to same output file as listfile uses above
  echo ![fqualify(_wh_tok)] >prntlf
else
  setvar hpcierr -1
endif
if hpcierr = 0 then
  xeq !hpfile !_wh_tok entry=process_listf <prntlf
endif
endwhile
deletevar _wh_@
return
... continued. . .

```

where (cont)

```
...
elseif "!entry" = "process_udcs" then
  # input redirected from the output of showcatalog
  setvar _wh_udcf rtrim(input())
  setvar _wh_eof finfo(hpstdin,"eof") -1
  while setvar(_wh_eof,_wh_eof-1) >= 0 do
    if lft(setvar(_wh_rec,rtrim(input())),1) = " " then
      # a UDC command name line
      if pmatch(ups(_wh_cmd),setvar(_wh_tok,word(_wh_rec))) then
        # display: UDC_command_name  UDC_level  UDC_filename
        echo !_wh_tok ![rpt(" ",26-len(_wh_tok))] &
          ![setvar(_wh_tok2,word(_wh_rec,,1))+rpt(" ",7-len(_wh_tok2))] &
          UDC in !_wh_udcf
      endif
    else
      # a UDC filename line
      setvar _wh_udcf _wh_rec
    endif
  endwhile
return
```

where (cont)

```

...
elseif "!entry" = "process_listf" then
  # input redirected from the output of listfile,6 or a simple filename
  setvar _wh_eof finfo(hpstdin,'eof')
  while setvar(_wh_eof,_wh_eof-1) >= 0 do
    setvar _wh_fc ""
    if setvar(_wh_fc, finfo(setvar(_wh_tok,ltrim(rtrim(input()))),'fmtfcode')) = ""
      setvar _wh_fc 'script'
    elseif _wh_fc <> 'NMPRG' and _wh_fc <> 'PROG' then
      setvar _wh_fc ""
    endif
    if _wh_fc <> "" and finfo(_wh_tok,'eof') > 0 then
      setvar _wh_lnk ""
      if _wh_fc = "script" and finfo(_wh_tok,'filetype') = 'SYMLINK' then
        setvar _wh_fc 'symlink'
        # get target of the symlink
        file lf7tmp;msg
        continue
        listfile !_wh_tok,7 >*lf7tmp
        if hpcierr = 0 then
          # discard first 4 records
          input _wh_lnk <*lf7tmp
          input _wh_lnk <*lf7tmp
          input _wh_lnk <*lf7tmp
          input _wh_lnk <*lf7tmp
          input _wh_lnk <*lf7tmp
          setvar _wh_lnk "--!> " + word(_wh_lnk,,-1)
        endif
      endif
    endif
  endwhile
endif

```

hp e3000

strategy

where (cont)

```
...
    # display: qualified_filename file_code or "script" and link if any
    echo !_wh_tok ![rpt(" ",max(0,26-len(_wh_tok)))] !_wh_fc &
        ![rpt(" ",7-len(_wh_fc))] !_wh_lnk
    endif
endwhile
return
endif
```

- :where @sh@

SHOWME	USER	UDC in SYS52801.UDC.SYS
SH	SYSTEM	UDC in HPPXUDC.PUB.SYS
SH.PUB.VANCE	NMPRG	
SHOWVOL.PUB.VANCE	script	
BASHELP.PUB.SYS	PROG	
HSHELL.PUB.SYS	script	
PUSH.SCRIPTS.SYS		script
RSH.HPBIN.SYS	NMPRG	
SH.HPBIN.SYS	NMPRG	
/bin/csh		NMPRG
/bin/ksh		symlink --> /SYS/HPBIN/SH
/bin/remsh	symlink	--> /ENM/PUB/REMSH
/bin/rsh		symlink --> /SYS/HPBIN/RSH
/bin/sh		symlink --> /SYS/HPBIN/SH

stream UDC - overview

- STREAM
ANYPARM streamparms = ![“”]
OPTION nohelp, **recursion**
...
if main entry point then
 # initialize ...
 - if “jobq=” not specified then read job file for job “card”
 - if still no “jobq=” then read config file matching “[jobname,]user.acct”
 - stream job in HPSYSJQ (default) or derived job queue
 - clean up
else
 # alternate entries
 separate entry name from remaining arguments
 ...
 if entry is **read_jobcard** then read job file looking for “:JOB”, concatenate
 continuation lines (&) and remove user.acct passwords
 ...
 elseif entry is **read_config** then
 read config file, match on “[jobname,]user.acct”
 ...
endif

stream UDC - “main”

```
# comments ...
if "!streamparms" = "" or pos("entry=", "!streamparms") = 0 then
  # main entry point of UDC
  setvar _str_jobfile word("!streamparms")           # extract 1st arg
  ...
  # extract remaining stream parameters
  setvar _str_parms ups( &
    repl(rht("!streamparms",-delimpos("!streamparms")), " ", ""))
  if setvar(_str_pos, pos(";JOBQ=", _str_parms)) > 0 then
    setvar _str_jobq word(_str_parms,,2,,_str_pos+5)
  endif
  if _str_jobq = "" then
    # no jobq=name in stream command so look at JOB "card"
    STREAM _str_jobcard entry=read_jobcard <! _str_jobfile
    if setvar(_str_pos, pos(";JOBQ=", _str_jobcard)) > 0 then
      setvar _str_jobq word(_str_jobcard,,2,,_str_pos+5)
    endif
  endif
endif
```

stream UDC - “main” (cont)

```

if _str_jobq = " and finfo(_str_config_file,'exists') then
    # No jobq=name specified so far so use the config file.
    STREAM ![word(_str_jobcard,";")] _str_jobq entry=read_config &
    <! _str_config_file
    if _str_jobq <> " then
        # found a match in config file, append jobq name to stream command line
        setvar _str_parms _str_parms + ";jobq=!_str_jobq"
    endif
endif
...
# now finally stream the job.
if _str_jobq = " then
    echo Job file "!_str_jobfile" streamed in default "HPSYSJQ" job queue.
else
    echo Job file "!_str_jobfile" streamed in "!_str_jobq" job queue.
endif
option norecursion
continue
stream !_str_jobfile !_str_parms
...

```

stream UDC - “read_jobcard”

else

alternate entry points for UDC.

setvar _str_entry word("!streamparms",,-1)

remove entry=name from parm line

setvar _str_entry_parms lft('!streamparms',pos('entry=','!streamparms')-1)

if _str_entry = "read_jobcard" then

Arg 1 is the *name* of the var to hold all of the JOB card right of "JOB".

Input redirected to the target job file being streamed

Read file until JOB card is found. Return, via arg1, this record,

including continuation lines, but less the "JOB" token itself. Remove

all passwords, if any. Skip leading comments in job file.

setvar _str_arg1 word(_str_entry_parms)

while str(setvar(!_str_arg1,ups(**input()**)),2,4) <> "JOB " **do**

endwhile

remove line numbers, if appropriate

if setvar(_str_numbered, numeric(rht(!_str_arg1,8))) **then**

setvar !_str_arg1 lft(!_str_arg1,len(!_str_arg1)-8)

endif

...

stream UDC - “read_jobcard” (cont)

```
...
# concatenate continuation (&) lines
while rht(setvar(!_str_arg1,rtrim(!_str_arg1)),1) = '&' do
  # remove & and read next input record
  setvar !_str_arg1 lft(!_str_arg1,len(!_str_arg1)-1)+ltrim(rht(input(), -2))
  if _str_numbered then
    setvar !_str_arg1 lft(!_str_arg1,len(!_str_arg1)-8)
  endif
endwhile
# remove passwords, if any
while setvar(_str_pos,pos('/',!_str_arg1)) > 0 do
  setvar !_str_arg1 repl(!_str_arg1,"/"+word(!_str_arg1,'.',',',,,,_str_pos+1), "")
endwhile
# return, upshifted, all args right of "JOB", and strip all blanks.
setvar !_str_arg1 ups(repl(xword(!_str_arg1)," ", ""))
return
```

stream UDC - “read_config”

elseif _str_entry = "read_config" then

Arg 1 is the "[jobname,]user.acct" name from the job card.

Arg 2 is the *name* of the var to return the jobQ name if the acct name

Input redirected to the jobQ config file.

setvar _str_arg1 word(_str_entry_parms, " ")

setvar _str_arg2 word(_str_entry_parms, " ", 2)

setvar _str_eof finfo (hpstdin, "eof")

...

read config file and find [jobname,]user.acct match (wildcards are ok)

while **setvar(_str_eof, _str_eof-1) >= 0** and &

(setvar(_str_rec, ltrim(rtrim(input())))) = "" or &

lft(_str_rec, 1) = '#' or &

not **pmatch**(ups(word(_str_rec, -2)), _str_ua) or &

(pos(',', _str_rec) > 0 and lft(_str_rec, 2) <> '@,' and &

not **pmatch**(ups(word(_str_rec)), _str_jname))) do

endwhile

if _str_eof >= 0 then

[jobname,]user.acct match, return jobq name

setvar !_str_arg2 word(_str_rec, -1)

endif

return

hp e3000

appendix

strategy

- COMMAND vs. HPCICOMMAND intrinsics
- i/o redirection basics
- more on expressions
- more on CI variables, including variable “arrays”
- more on UDCs and scripts
 - file layouts
 - feature comparisons
 - performance considerations
 - parameters
- more examples

COMMAND intrinsic

- COMMAND is a programmatic system call (intrinsic)
syntax: COMMAND (cmdimage, error, parm)
- implemented in native mode (NM, PA-RISC mode)
- use COMMAND for system level services, like:
 - building, altering, copying purging a file
- no UDC search (a UDC cannot intercept “cmdimage”)
- no command file or implied program file search
- returns command error number and error location
(for positive parmnum), or file system error number for negative parmnum

HPCICOMMAND intrinsic

- HPCICOMMAND is an intrinsic
syntax: HPCICOMMAND (cmdimage,error,parm
[,msglevel])
- implemented in native mode (NM, PA-RISC mode)
- use HPCICOMMAND for a “window” to the CI, e.g.:
 - providing a command interface to a program, “:cmdname”
- UDCs searched first
- command file and implied program files searched
- returns command error number and error location or file system error number.
- Msglevel controls CI errors/warnings -- similar to the HPMSGFENCE variable

CI i/o redirection

strategy

- > name - redirect output from \$STDLIST to “name”
 - “name” will be overwritten if it already exists
 - file will be saved as “name”;rec=-256,,v,ascii;disc=10000;TEMP
 - file name can be MPE or POSIX syntax
- >> name - redirect, append output from \$STDLIST to “name”
 - same file attributes for “name” if it is created
- < name - redirect input from \$STDIN to “name”
 - “name” must exist (TEMP files looked for before PERM files)
- I/O redirection has no meaning if the command does not do I/O to \$STDIN or \$STDLIST
- available on all commands, except:
 - IF, ELSEIF, SETVAR, CALC, WHILE, COMMENT, SETJCW, TELL, TELLOP, WARN, REMOTE.

CI i/o redirection (cont)

strategy

- how it works:
 - CI ensures the command is not one of the excluded commands
 - CI scans the command line looking for <, >, >> followed by a possible filename (after explicit variable resolution has already occurred)
 - text inside quotes is excluded from this scan
 - text inside square brackets is excluded from the scan
 - filename is opened and “exchanged” for the \$STDIN or \$STDLIST
 - after the command completes the redirection is undone
- examples:
 - INPUT varname < filename
 - ECHO The next answer is: !result >>filename
 - LISTFILE ./@,6 > filename
 - PURGEACCT myacct <Yesfile
 - PURGE foo@ ;temp ;noconfirm >\$null
 - ECHO You need to include !<THIS!> too!

CI expressions

- operators:
 - + (ints and strings), -, *, /, ^, (), <, <=, >, >=, =, AND, BAND, BNOT, BOR, BXOR, CSL, CSR, LSL, LSR, MOD, NOT, OR, XOR
- precedence (high to low):
 - 1) variable dereferencing
 - 2) unary + or -
 - 3) bit operators (csr, lsl...)
 - 4) exponentiation (^)
 - 5) *, /, mod
 - 6) +, -
 - 7) <, <=, =, >, >=
 - 8) logical operators (not, or...)
 - left to right evaluation, except exponentiation is r-to-l

CI variables

- 113 predefined “HP” variables
- user can create their own variables via :SETVAR
- variable types are: integer (signed 32 bits), Boolean and string (up 1024 characters)
- variable names can be up 255 alphanumeric alphanumeric and “_” (cannot start with number)
- predefined variable cannot be deleted, some allow write access
 - :SHOWVAR @ ; HP -- shows all predefined variables
- can see user defined variables for another job/session (need SM)
 - :SHOWVAR @ ; job=#S or Jnnn
- the bound() function returns true if the named variable exists
- variables deleted when job / session terminates
- :HELP variables and :HELP VariableName

compound variables

strategy

- :setvar a "!!b" # B is not referenced, 2!'s fold to 1
- :setvar b "123"
- :showvar a, b A=!b B=123
- :echo b is !b, a is !a b is 123, a is 123
- :setvar a123 "xyz"
- :echo Compound var "a!!b": !"a!b" Compound var "a!b": xyz
- :setvar J 2
- :setvar VAL2 "bar"
- :setvar VAL3 "foo"
 - :calc VAL!J bar
 - :calc VAL![J] bar
 - :calc VAL![decimal(J)] bar
 - :calc VAL![setvar(J,J+1)] foo

variables arrays

- simple convention using standard CI variables
- varname0 = number of elements in the array
varname1...varnameN = array elements, 1 .. !varname0
varname!J = name of element J
!"varname!J" = value of element J

- :showvar buffer@

```
BUFFER0 = 6  
BUFFER1 = aaa  
BUFFER2 = bbb  
BUFFER3 = ccc  
BUFFER4 = ddd  
BUFFER5 = eee  
BUFFER6 = fff
```

variable array example

- centering output:

```

PARM count=5
setvar cnt 0
while setvar(cnt,cnt+1) <= !count do
    setvar string!cnt,input("Enter string !cnt: ")
endwhile
setvar cnt 0
while setvar(cnt,cnt+1) <= !count do
    echo ![rpt(" ",39-len(string!cnt))]"string!cnt"
endwhile

```

“Center” script

:center

```

Enter string 1: The great thing about Open Source
Enter string 2: software is that you can
Enter string 3: have any color
Enter string 4: "screen of death"
Enter string 5: that you want.

```

```

    The great thing about Open Source
        software is that you can
            have any color
                "screen of death"
                    that you want.

```

filling variables arrays -- wrong!

- example 1: # array name is "rec"

```
setvar j 0
setvar looping true
while looping do
  input name, "Enter name "
  if name = "" then
    setvar looping false
  else
    setvar j j+1
    setvar rec!j name
  endif
endwhile
setvar rec0 j
```
- :exmpl1
 - infinite loop!, won't end until <break>

filling variables arrays (cont)

- example 2:

```
setvar j 0
setvar looping true
while looping do
    setvar NAME ""
    input name, "Enter name "
    if name = "" then
        setvar looping false
    else
        setvar j j+1
        setvar rec!j name
    endif
endwhile
setvar rec0 j
```
- :exmpl2 <datafile (datafile has 20 text records)
("enter name" prompt shown 20 times snipped...)
End of file on input. (CIERR 900)
input name, "enter name "
Error executing commands in WHILE loop. (CIERR 10310)

filling variables arrays (cont)

- example 3;

```

setvar j 0
if HPINTERACTIVE then
    setvar prompt "Name = "
    setvar limit 2^30
    setvar test 'name= " '
else
    setvar prompt ""
    setvar limit FINFO (HPSTDIN, "eof")
    setvar test "false"
endif
while (j < limit) do
    setvar name ""
    input name , !prompt
    if !test then
        setvar limit 0                # exit interactive input
    else
        setvar j j+1
        setvar rec!j name
    endif
endwhile
setvar rec0 j

```

filling variables arrays (cont)

- :exmpl3 <datafile
- :showvar rec@
REC1 = line1
REC2 = line2
...
REC20 = line20
REC0 = 20
- performance:
 - Script as is: 100 records: 530 millisecs
 - Script modified for file input only:
100 records: 380 millisecs

filling variables arrays (cont)

strategy

- can we fill arrays (and read files) faster?
- example 4:

```
setvar rec0 0
setvar limit FINFO (HPSTDIN, "eof")
while setvar(rec0, rec0+1) <= limit and &
    setvar(rec![rec0+1], input()) <> chr(1) do
endwhile
setvar rec0 rec0-1
```

- performance (:exmpl4 <datafile):
 - 100 records: **185 millisecs** (twice as fast!)

hp e3000

predefined variables

strategy

- HPAUTOCONT - set TRUE causes CI to behave as if each command is protected by a :continue.
- HPCMDTRACE - set TRUE causes UDC / scripts to echo each command line as long as OPTION NOHELP not specified. Useful for debugging.
- HPCPUMSECS - tracks the number of milliseconds of CPU time used by the process. useful for measuring script performance.
- HPCWD - current working directory in POSIX syntax.
- HPDATETIME - contains the date/time in CenturyYearMonthDateHourMinuteSecondMicrosecond format.
- HPDOY - the day number of the year from 1..365.
- HPFILE - the name of the executing script or UDC file.
- HPINTERACTIVE - TRUE means \$STDIN and \$STDLIST do not form an interactive pair, useful to test if it is ok to prompt the user.
- HPLASTJOB - the job ID of the job you most recently streamed, useful for a default parm value in UDCs that alter priority, show processes, etc.

predefined variables (cont)

- HPLASTSPID - the \$STDLIST spoolfile ID of the last job streamed, useful in
:print !hplastspid.out.hpspool
- HPLOCIPADDR - IP address for your system.
- HPMAXPIN - the maximum number of processes supported on your system.
- HPPATH - list of group[.acct] or directory names used to search for script and program files
- HPPIN - the Process Identification Number (PIN) for the current process.
- HPPROMPT - the CI's command prompt, useful to contain other info like: !!HPCWD, !!HPCMDNUM, !!HPGROUP, etc.
- HPSPoolID - the \$STDLIST spoolfile ID -- if executing in a job.
- HPSTDIN - the filename for \$STDIN, useful in script "subroutines" where input has been redirected to a disk file
- HPSTREAMEDBY - the "Jobname,User.Acct (jobIDnum)" of the job/session that streamed the current job.
- HPUSERCAPF - formatted user capabilities, useful to test if user has desired capability, e.g. if pos("SM",hpusercapf) > 0 then

UDCs

- user defined command files (UDCs) - a single file that contains 1 or more command definitions, separated by a row of asterisks (***)
- features:
 - simple way to execute several commands via one command
 - allow built-in MPE commands to be overridden
 - can be invoked each time the user logs on
 - require lock and (read or eXecute) access to the file
 - cataloged (defined to the system) for easy viewing and prevention of accidental deletion -- see :SETCATALOG and :SHOWCATALOG commands
 - can be defined for each user or account or at the system level
 - more difficult to modify since file usually opened by users

command files (scripts)

- command file - a file that contains a single command definition
- features:
 - same convenience as UDCs
 - searched for after UDCs and built-in commands using the HPPATH variable -- default HPPATH includes "logon group, PUB.logon account, PUB.SYS, ARPA.SYS"
 - require read or eXecute access
 - easy to modify since file is only in use while it is being executed
 - very similar to unix scripts or DOS bat files

UDC / script comparisons

- similarities:
 - ASCII, NOCCTL, numbered or unnumbered, max 511 record width
 - optional parameter line ok - max of 255 arguments
 - optional options, e.g. HELP, NOBREAK, RECURSION
 - optional body (actual commands)
 - no inline data, unlike Unix 'here' files :(
 - can protect file contents by allowing eXecute access-only security, i.e., denying read access

UDC / script comparisons (cont)

- differences:
 - scripts can be variable record width files
 - UDCs require lock access, scripts don't
 - script names can be in POSIX syntax, UDC filenames must be in MPE syntax
 - UDC name cannot exceed 16 chars, script name length follows rules for MPE and POSIX named files
 - EOF for a script is the real eof, end of a UDC command is one or more asterisks, starting in column 1

UDC file layout

strategy

filename: AUDC.PUB.SYS

header:

```
UDCcommandname [ parm1] [ p2 [= value ] ]  
[ ANYPARM parm4 [= value] ]  
[ OPTION option_list ]
```

body:

any MPE command, UDC or script
(option list or option recursion supported in body too)

end-of-UDC

***** (end of this command definition)

header:

```
NextUDCcommand [ parm1 ]  
[ PARM P2, P3 = value ]
```

body:

```
[ OPTION option_list ]  
any MPE command etc...
```

script file layout

strategy

filename: PRNT.SCRIPTS.SYS

header: [**PARM** parm1, parm2 [= value]]
[**ANYPARM** parm3 [= value]]
[**OPTION** option_list]

body: any MPE command, UDC or script
(:option list or :option recursion supported in body too)

eof

filename: LG.SCRIPTS.SYS

header: PARM ...
OPTION nohelp ...

body: any MPE command etc...

UDC search order

strategy

UDCUSER.udc.finance

1. Invoke UDCC, which calls UDCA with the argument "ghi"
2. UDCA is found, starting after the UDCC definition (option NOrecursion default)
3. The line "p1=ghi" is echoed
4. Invoke UDCB, which calls UDCA passing the arg "def". The recursion option causes the first UDCA to be found. This calls UDCC and follows the path at step 1 above
5. The line "p1=def" is echoed

File:

```
UDCA p1 = abc ←
option NOrecursion
udcC !p1
***
```

```
UDCB p1 = def
option recursion
udcA !p1
***
```

```
UDCC p1 = ghi
udcA !p1
***
```

```
UDCA p1 = xyz ←
echo p1=!p1
***
```

script search order

- scripts and programs searched for after command is known to not be a UDC and to not be a built-in command
- same order for scripts and for program files
- fully or partially qualified names are executed without qualification
- unqualified names are combined with HPPATH elements to form qualified filenames:
 - first match is executed
 - filecode = 1029, 1030 for program files
 - EOF > 0 and filecode in 0..1023 for script files
 - to execute POSIX named scripts a POSIX named directory must be present in the HPPATH variable

UDCs vs. scripts

- option logon
 - UDCs only (a script can be executed from an “option logon” UDC)
 - logon UDCs executed in this order:
 - 1. System level 2. Account level 3. User level
(opposite of the non-logon execution order!)
- CI command search order:
 - A. UDCs (1. User level 2. Account level 3. System level)
 - thus UDCs can override built-in commands
 - B. built-in MPE commands, e.g. LISTFILE
 - C. script and program files. HPPATH variable used to qualify unqualified filenames
 - :XEQ command allows script to be same name as UDC or built-in command, e.g. :xeq listf.scripts.sys

UDCs vs. scripts (cont.)

strategy

- performance

- logon time:

- 9 UDC files, 379 UDCs, 6050 lines: 1/2 sec.

most overhead in opening and cataloging the UDC files

- to make logons faster remove unneeded UDCs

- execution time:

- identical (within 1 msec) for simple UDCs vs scripts,
however:

- factorial script:

- :fac 12 157 msec

- factorial UDC (option recursion):

- :facudc 12 100 msec

- file close logging impacts performance for scripts more since they
are opened/closed for each invocation

UDCs vs. scripts (cont.)

- maintenance / flexibility / security
 - SETCATALOG opens UDC file, cannot edit without un-cataloging file, but difficult to accidentally purge UDC file
 - UDC commands grouped together in same file, easier to view and organize
 - UDC file can be lockword protected but users don't need to know lockword to execute a UDC
-
- scripts opened while being executed (no cataloging), can be purged and edited more easily than UDCs
 - scripts can live anywhere on system. Convention is to place general scripts in a common location that grants read or eXecute access to all, e.g. "XEQ.SYS" group
 - if script protected by lockword then it must be supplied each time the script is executed

UDC / script exit

- EOF -- real EOF for scripts, a row of asterisks (starting in column 1) for UDCs
- :BYE, :EOJ, :EXIT -- terminate the CI too, to use BYE or EOJ must be the root CI
- :RETURN -- useful for entry point exit, error handling, help text - jumps back one call level
- :ESCAPE -- useful to jump all the back to the CI, or an active :CONTINUE. In a job without a :CONTINUE, :escape terminates the job. Sessions are not terminated by :escape. Can optionally set CIERROR and HPCIERR variables to an error number

parameters

- syntax: ParmName [= value]
 - supplying a value means the parameter is optional. If no value is defined the parameter is considered required.
 - max parm name is 255 bytes, chars A-Z, 0-9, “_”
 - max parm value is limited by the CI's command buffer size (currently 511 characters)
 - all parm values are un-typed, regardless of quoting
 - Parms are separated by a space, comma or semicolon
 - default value may be a: number, string, !variable, ![expression], an earlier defined parm (!parm)
 - all parameters must be explicitly referenced in the UDC/script body, e.g. !parmname
 - the scope of a parm is the body of the UDC/script

parameters (cont)

- all parameters are passed “by value”, meaning the parm value cannot be changed within the UDC/script
- a parm value can be the name of a CI variable, thus it is possible for a UDC/script to accept a variable name, via a parm, and modify that variable's value, e.g.

```
SUM a, b, result_var
setvar !result_var !a + !b
*****
```

SUM is a UDC name

```
:SUM 10, 2^10, x
:showvar x
```

X = 1034

```
:setvar I 10
:setvar J 12
:SUM i, j, x
:showvar x
```

inside SUM: setvar x, i + j
X = 22

ANYPARM parameter

- all delimiters ignored
- must be last parameter defined in UDC/script
- only one ANYPARM allowed
- only way to capture user entered delimiters, without requiring user to quote everything
- example:

```
TELLT user
ANYPARM msg = ""
# prepends timestamp and highlights msg text
tell !user; at !hptimef: ![chr(27)]&dB !msg
```

```
:TELLT op.sys Hi, what's up,,,, system seems fast!
FROM S68 JEFF.UI/3:27 PM: Hi, what's up,,,, system
seems...
```

- anyparm() function is useful with ANYPARM parameters

brief acct, group, user, dir listings

- LG, LU, LA and LD scripts:
 - PARM group=@ "LG"
listgroup !group; format=brief

 - PARM user=@ "LU"
listuser !user; format=brief

 - PARM acct=@ "LA"
listacct !acct; format=brief

 - PARM dir=./@ "LD"
setvar _dir "!dir"
if delimpos(_dir, "./") <> 1 then
convert MPE name to POSIX name
setvar _dir dirname(fqualify(_dir)) + "/" + basename(_dir)
endif
listfile !_dir, 6; seleq=[object=HFSDIR]; tree

CI grep

- PARM pattern, file, entry=main
This script implements unix `$grep -in <pattern> <file>`.
setvar savecpu hpcpumsecs
if '!entry' = 'main' then
 errclear
 setvar _grep_matches 0
 if not finfo('!file','exists') then
 echo File "!file" not found.
 return
endif
continue
xeq !HPFILE !pattern !file entry=read_match <!file
echo ![hpcpumsecs-savecpu] msec ...
echo !_grep_eof records read -- !_grep_matches lines match "!pattern"
deletevar _grep_
return
... (continued on next slide)

CI grep (cont)

```

elseif '!entry' = 'read_match' then
  # finds each "pattern" in "file" and echoes the record + line num
  # input redirected to "!file"
  setvar _grep_eof finfo("!file","eof")
  setvar _grep_recno 0
  setvar _grep_pat ups("!pattern")
  while setvar(_grep_recno,_grep_recno+1) <= _grep_eof and &
    setvar(_grep_rec, rtrim(input())) <> chr(1) do
    if pos(_grep_pat,ups(_grep_rec)) > 0 then
      echo !_grep_recno) !_grep_rec
      setvar _grep_matches _grep_matches+1
    endif
  endwhile
  return
endif

```

- 4667 msec ...
1669 records read -- 18 lines match "version"
- 4627 msec ...
1669 records read -- 0 lines match "foo"