

# 6.0 CI Enhancements

and other improvements presented by Jeff Vance, HP-CSY

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

# Outline

## • New/modified CI commands

- ABORTPROC, LISTF, PAUSE, SHOWVAR, COPY, User-Defined Job Queues , UDCs

## • New/modified CI functions and variables

- jinfo(), jobcnt(), finfo(), fsyntax(), fqualify(),HPDATETIME, HPDOY, HPLEAPYEAR...

## • New/modified Intrinsic

- HPCALENDAR, HPDATECONVERT, FLABELINFO...

## • POSIX enhancements

- better installation, expanded filenames ...

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

- 6.0 Powerpatch 2 (PP2):
  - APACHE and LDAP in FOS
  - :ABORTPROC command (tentative)
  - User defined job queue fixes
- 6.0 PP1:
  - jinfo, jobcnt,wordcnt CI functions
  - HPDATETIME, HPDOY, HPLEAPYEAR, HPHHMMSSMMM, HPYYYYMMDD CI variables

- 6.0 base:
  - JAVA, SAMBA, DNS in FOS
  - User Defined Job Queues
  - :SHOWVAR ;JOB= enhancement
  - HPMSGFENCE variable enhancement
  - POSIX improvements: install fixes, /etc/profile.local, libc bug fixes (sigsetjmp, siglongjmp), shared libs created, socket fixes for BIND, tar -h enhancement, re-entrant name service routines (gethostbyname\_r, gethostbyaddr\_r), poll() support, larger shell command buffer.
  
- 5.5 PP5, PP4, PP3:
  - :COPY to a directory (pp5)
  - extended POSIX filename support (pp5)
  - anyparm, basename, decimal, dirname, finfo, fsyntax, fqualify, xword functions (pp5)
  - HPDATECONVERT, HPDATEFORMAT, HPDATEDIFF, HPDATEOFFSET, HPDATEVALIDATE, HPCALENDAR, HPFMTCALENDAR intrinsics (pp4)
  - :LISTF access , :PAUSE ;JOB=, :PRINT ;nonum, :INPUT ;READCNT= (pp3)
  - HPSPOOLID, HPLASTSPID CI variables (pp3)
  
- Volume Mgmt UDCs (NEWACCT, NEWGROUP, PURGEACCT, PURGEGROUP) on jazz ([jazz.external.hp.com/src/scripts/udcvol](http://jazz.external.hp.com/src/scripts/udcvol))

# ABORTPROC command

- **ABORTPROC pin | (pin1, pin2, , pinN) [ ;SYSTEM ]**
- **requires SM or OP capabilities**
- **warns if abort is already pending**
- **works for individual threads (pin.TID syntax)**
- **terminates children of target process**
- **allows aborting system processes ONLY if they are detached and ;SYSTEM specified and SM cap**

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

The ABORTPROC command is planned for 6.0 pp2. It accepts one or more Process IDs (PINs), using the same format as SHOWPROC and ALTPROC. An attempt is made to terminate each specified PIN. ABORTPROC is not more powerful than the ABORTJOB command, in fact, it calls the same internal routines as ABORTJOB. If ABORTJOB cannot abort a job or session due to a hung process, it is unlikely that ABORTPROC will do any better. However, the potential now exists for single processes to be killed without aborting the entire job or session structure.

A process cannot be killed if it is critical. Processes set themselves critical when they wish to defer aborts. This is a common method used in order to preserve data integrity, or to guarantee system resources, like semaphores or shared memory, are released before a process dies. ABORTPROC will not abort any critical processes. Typically a processes is marked critical for

short duration's, and ABORTPROC sets an abort pending flag, which will cause the process to terminate once it is no longer critical. If the target process remains critical it cannot be aborted by ABORTPROC.

If the target process has children processes, ABORTPROC also kills the children processes. Unlike most UNIX systems, MPE does not support the concept of children processes surviving the death of their parent.

System Managers will be able to kill system processes that have made themselves detached. This turns out to be just a few of the standard system processes. Examples include: some XM, diagnostics and Volume Management processes.

# LISTF access command

- LISTF fileset , 8 or 9

**LISTFILE fileset , access or locks**

**;SELEQ= [**

**ACCESS = inuse | open | excl | lock**

**CODE = number | mnemonic | PRIV ]**

- **:listfile @. @ , 8; seleq= [access= inuse]**

**FILE: HPPXUDC.PUB.SYS**

**15 Accessors(O:15,P:15,L:0,W:0,R:15),Share**

**#S265 MIKEP.HPE P:2,L:0,W:0,R:2 REM: 15.14.16.198 . . .**

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

- LISTFILE supports access or 8 and locks or 9.
- Format 8 (access) shows the qualified filename, an access summary line, one data line for each job/session accessing the file.

- Format 8 info: job/session number, user.acct name, number of processes accessing, locking, writing, reading the file. The last item displayed is the LDEV of a local session accessor, or the IP address of a remote session accessor, or the spoolfile ID for a local job accessor.
- Format 9 (locks) shows all of format 8 info plus: PIN, program name, access method, sharing mode (e.g. gmulti, ear, etc), record number, file number, locks the process owns or is impeded on.
- Restrictions: to see individual accessor data for a given file you must have SM or OP cap, or be the AM and the file's GID matches your GID, or be the file's owner. Otherwise just the accessor summary line is displayed.
- There are additional checks to reveal the IP address and program name: SM, OP, ND, NA or PM cap needed to see IP address. SHOWPROC rules are enforced to see the program name: process is in your job/session structure, or process is in your logon acct and JOBSECURITY is low or user has OP or SM capabilities.
- The number of accessors (15 on this slide) represents all detectable accessors of the file. This includes processes that formally open the file (via FOPEN, HPFOPEN, POSIX open, etc.) and processes that open the file bypassing the file system by calling the internal sm\_open routine. This is the same number seen in LISTF,3 and returned by the new FINFO item accessors , and examined by the new INUSE SELEQ verb.

## LISTF access (cont.)

- **LISTFILE hppxudc.pub.sys, 9**

**5 Accessors(O:5, P:5, L5, W:0, R:5),Shar**

**#S263 JV,MGR.JVNM P:3,L:3,W:1,R:3 LDEV: 47**

**#P46 (LFCI.PUB.SYS)**

**ACCESS: R-excl REC#: 0 FNUM: 13**

**LOCKS--Owner-- --Waiter--**

**FLOCK**

**OPEN**

**#P86 (JSMINI.PUB.SYS)**

**ACCESS: R-excl REC#: 336 FNUM: 16**

**LOCKS--Owner-- --Waiter--**

**FLOCK . . .**

- **FINFO(filename, accessors )**

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

- The formal open count (O: in the accessor summary line) may be less than the total accessor count since some processes may have opened the file but did not call a standard file open API. STORE is such a process. Processes that do not formally open the file are usually not detected by LISTF. Only if this process also locks one of the semaphores mentioned below can LISTF determine some limited data about the accessor.
- Three file systems locks are evaluated: 1) FLOCK semaphore used by the FLOCK intrinsic and by directory services, 2) OPEN semaphore used by the file system to protect several data structures, 3) GUFD semaphore used in file I/O. A process can lock one or more of these semaphores.
- The OPEN semaphore can be shared and thus can have more than one owner. The FLOCK and GUFD semaphores are locked exclusively so they will have only one owner but may have several processes waiting.
- It is possible for a process to lock one of these semaphores but not open the file at all. The CI does this in LISTF, and callers of AIFSYSWIDEGET item 2065 (like MPEX) are another example. In these cases the reported access count is lower than the number of accessors displayed. As is the case for processes using sm\_open, only limited data can be displayed for these phantom accessors.

# PAUSE command

- **PAUSE [seconds] [JOB= jobID] [interval] [; EXISTS | ;WAIT | ;NOTEXIST ]**
  - jobID - [#]J | Snnnn, @, @J, @S, [@J|S:] [jobname,]user.acct
- - EXISTS - pause while the target job(s) exist
  - WAIT - pause while the job(s) are waiting
  - NOTEXIST - pause while the job does not exist

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

- 
- Pausing for a job works by efficiently polling the JMAT. The interval is computed based on the number of times the same job has been polled, favoring shorter jobs. The interval parameter allows static polling. Long poll periods risk missing job state transitions, which mostly matters with the NOTEXIST option.
  - EXISTS is the default when JOB= is specified. As long as one or more jobs matching JOB= exist in any state (intro, wait, susp, scheduled, exec) the pause continues.
  - WAIT tells pause to sleep as long as the job matching JOB= is waiting. If the job starts executing, terminates, is aborted, etc. the pause is over.
  - NOTEXIST tells pause to sleep while there are no jobs matching JOB=. Once a job is launched that matches JOB= the pause terminates.
  - To pause for X seconds or until job Y terminates, whichever is shortest, enter: :pause x ; job=Y A

CI warning is reported if job Y still exists after X seconds.

- To pause for X seconds or until job Y completes, whichever is longest use two commands: :  
pause x : pause job=Y
- To select jobs without a job name use a null jobname ( , ) for the job name. To select only jobs with a job name specify the jobname parm. If the job name doesn't matter omit it. Jobname, username and acctname can be wildcarded.
- To select jobs that exactly match you , enter: pause job= @!hpjobtype:!hpjobname,!hpuser.!  
hpaccount

# PAUSE - examples

- **PAUSE job=#J123**

sleeps until job J123 terminates

- **PAUSE job=!HPLASTJOB ;wait**

sleeps until the job just streamed (hplastjob variable) is no longer waiting: it could be done or still executing

- **PAUSE ![60\*60], @J**

sleeps until all jobs on the system have finished or 1 hour whichever is shortest. CIWARN  
If jobs exists after 1 hr.

- **PAUSE job= @J:@, @.PROD@**

sleeps until all jobs with job names in all accounts beginning with PROD have terminated (quotes required)

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

- The jobID value must be quoted if a jobname is included since the jobname must be followed by a comma, and a comma is a command token delimiter.
- The @J: and @S: syntax is new in 6.0 PP1. This allows only jobs or only sessions matching a particular [jobname,]user.acct to be selected. Previously, both jobs and sessions would qualify if they matched the [jobname,]user.acct specification.

- The NOTEXIST option must be used somewhat carefully, especially if a single job is specified.

For example:

```
pause job=#j121 ;notexist
```

Since pause polls it is possible that job J121 is launched and terminates before the pause command does its next poll. In this event the pause will continue forever until break is hit. Additionally, the next assigned job/session number is not predictable on a busy system. It is probably more useful to use wildcards in conjunction with NOTEXIST. E.g.:

```
pause job=@J ;notexist
```

which sleeps until the first job is detected. It is also reasonable to use a maximum limit on the pause duration whenever NOTEXIST is also declared. E.g.:

```
pause 1800 ;job= backup,opsys.sys ;notexist
```

sleeps until the backup job has started or for 30 minutes, whichever occurs first.

# SHOWVAR command

- **SHOWVAR [var 1] [,var 2] [,var N]**

- **[ JOB= jobID [ ;USER | HP | ANY ]**

- jobID - [#]J | Snnn
    - USER - show only user-defined vars HP - show only HP predefined vars ANY - show user or predefined vars

- **Need SM to specify JOB=**

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

- If no variable name is specified @ and USER are assumed. If a variable name is specified ANY becomes the default. However, when JOB= is used it always implies USER regardless of the variable name or other options.
- It is now possible to define several variables beginning with the letter H and easily view them. E.g.:  

```
setvar HELP_TEXT . setvar HELP_PGM . SHOWVAR H@ ;USER
```
- To see all user-defined variables associated with job #J123, enter:  

```
SHOWVAR ;job= J123 ;user --or-- SHOWVAR @ ;job= #J123; user
```

Note: it is be a good idea to explicitly specify USER, as above, since it may be possible in future releases to see the predefined variables of another job.

- To see all predefined HP variables containing CAP , enter:

`SHOWVAR @cap@ ;HP`

This will omit any user-defined variables that also contain cap in their name.

# COPY command

- **COPY from\_file, to\_file [ ;ask | yes | no ]**

**TO\_FILE** - can be a filename, group name and now a directory name.

- **A space can separate the from= and to= names.**
- **Examples: 1. COPY abc .GROUP.acct 2. COPY abc /ACCT/ GROUP 3. COPY ./abc /DIRname 4. COPY abc ./dirName/ 5. COPY abc DIRNAME**

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

- **COPY to\_file algorithm:**
  - if to\_file is omitted, the base filename portion of the from\_file becomes the to\_file name. This to\_file name is unqualified, and one of the rules below will be applied.
  - if to\_file syntax is .group the from\_file is copied to the specified group.
  - If the to\_file is a POSIX name and ends in a slash(/) it is assumed (but not verified) to be a directory name and the from\_file is copied to that directory.
  - If to\_file exists and is a directory, the from\_file is copied to that directory. Note: the directory name can be in MPE or POSIX syntax.
  - If to\_file exists and names a file, the user is prompted to purge it before it is overwritten.
  - If to\_file does not exist from\_file is copied to to\_file .
- 1. abc --> ABC.GROUP.ACCT

- 2. abc --> /ACCT/GROUP/ABC
- 3. ./abc --> /DIRname/abc
- 4. abc --> CWD/dirName/ABC
- 5. abc --> CWD/DIRNAME/ABC
- It is a good idea in scripts and JCL to use a trailing slash (/) when the to\_file is really a directory name. This improves readability and should make future support easier.

# User-Defined Job Queues

- **NEWJOBQ q\_name [;limit=n]**
- **PURGEJOBQ q\_name**
- **LISTJOBQ**
- **LIMIT [+ | -] joblim, seslim [;JOBQ=q\_name]**
- **JOB user.acct [;JOBQ=q\_name]**
- **ALTJOB [;JOBQ=q\_name] ;HIPRI**
- **STREAM [;JOBQ=q\_name]**
- **SHOWJOB [;JOBQ]**
- **STREAM UDC available on jazz**

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

- Multiple queues can be defined for jobs only - not for sessions. Job queue properties today are a name and a limit. A job queue name cannot exceed eight characters. Up to 255 job queues are supported. The single job queue we have now has been named HPSYSJQ and is the default job queue for all jobs not assigned to a queue.

- SM or OP capability is required to create a new job queue.
- Job queues survive a `START RECOVERY` but are destroyed by `START NORECOVERY`. It is recommended to create your job queues in the `SYSSTART` file; however, you will need to stream a job to do this since the User-defined Job Queues commands are not directly supported in `SYSSTART`.
- There is still a global system-wide job limit which is modified by the `LIMIT` command. That is, `LIMIT` without an explicit job queue name modifies the `HPSYSJQ` queue. The global limit overrides the limit for any individual job queues. So, if a specific job queue has more capacity but the global limit has been reached no jobs can begin execution in that job queue until: a) the global limit is raised, b) one or more jobs terminate and there are no competing jobs of higher priority.
- How the next job to launch is chosen: 1) the highest `INPRI` job across all job queues is selected first. 2) ties are broken by choosing the job with the earliest introduced time.
- Only empty job queues can be purged.
- `ALTJOB` can move a job from one queue to another, which requires `SM` or `OP`. When a job is moved the limit of the target queue is ignored. `ALTJOB` can also make a waiting job `HIPRI`, so it can immediately start executing.
- Job queue precedence: `:STREAM... jobq=` overrides `:JOB... jobq=` which overrides the `HPSYSJQ` system defined default job queue.

# Volume Mgmt UDCs

- **Simple config file used to map accounts to volume sets. E. g. @PROD@ ---> PRODUCT\_VOLSET**
- **NEWACCT - don t need ONVS=**
- **NEWGROUP - don t need ONVS= or HOMEVS=**
- **PURGEACCT - don t need ONVS= for each volume set the account resides**
- **PURGEGROUP - don t need ONVS=, and supports wildcarded group names**
- **supporting scripts to build the config file and to check volume set usage**

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

- See [jazz.external.hp.com/src/scripts/udcvol](http://jazz.external.hp.com/src/scripts/udcvol) to download these UDCs and documentation.
- Default configuration file name is ACCTUVOL.pub.sys. Each entry in this file maps a group.acct or just an account to a particular user volume set. The group and account names can be wildcarded.

- NEWGROUP volume set precedence rules: 1) ONVS= in the command line has precedence over 2) a group.acct entry in the config file, which has precedence over 3) an account name in the config file, which has precedence over 4) a volume set name prompted for by the UDC, which has precedence over 5) the system volume set (MPEXL\_SYSTEM\_VOLUME\_SET).
- It is easy for a group to inherit the volume set of its parent account -- only requires an account entry in the configuration (ACCTUVOL) file.
- If the new group does not have an account on the target volume set the account will be created as long as the user has SM.
- NEWACCT has the same precedence rules as NEWGROUP, except that group.acct names are not searched for in the configuration file.
- FILES= for newacct and newgroup is handled such that no files are permitted on the system volume set -- all files will live on the user volume set. This is done automatically by the UDCs.
- The UDCs ensure that new accounts will have CV and UV capability, as will the account manager
- PURGEACCT detects all volume sets containing the account and lets you easily delete the account from each of these volume sets.
- PURGEGROUP deletes the group from the system volume set and the HOMEd to volume set. The group name can be wildcarded.

# New CI Functions

- **anyparm** - treats unquoted list of chars as a single string
- **basename** - returns the filename portion of an MPE or POSIX filename
- **dirname** - returns the POSIX-style directory components of an MPE or POSIX filename - name is not qualified first
- **fsyntax** - returns the syntax of an MPE or POSIX filename
- **fqualify** - returns a fully qualified MPE or POSIX filename
- **jinfo** - returns info specific to a job or session
- **jobcnt** - returns number of jobs matching
- **wordcnt** - returns number of tokens in input string
- **xword** - returns its input string less the word defined by its arguments - counterpart to the word() function

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

Online HELP available for all CI functions and variables.

- `anyparm(anything)` - all but last right parentheses are considered part of value ANYPARM p2  
`setvar infostr anyparm(!p2)`
- `basename(str [,suffix])` - removes suffix from basename if supplied  
`basename( aa.bb.cc ) = AA`  
`basename( /aa/bb/cc ) = cc`  
`basename( ./aa/bb.x , x ) = bb`
- `dirname(str)` - converts name to POSIX then extracts the directory name  
`dirname( aa.bb.cc ) = /`  
`CC/BB`  
`dirname( aa/bb/cc ) = /aa/bb`  
`dirname( abc ) = .` # not qualified  
`dirname(fqualify( abc )) = /`  
`ACCT/GRP --or-- /CWD`
- `fsyntax(str)` - supports wildcards,lockwords,\$null and errors  
`fsyntax( abc@ ) = MPE;WILD`  
`fsyntax( /abc ) = POSIX`  
`fsyntax( a.b.c.d ) = ERROR=426`
- `fqualify(str)` - tries to qualify name as MPE first then POSIX  
`fqualify( abc ) = ABC.GRP.ACCT --`  
`or-- /CWD/ABC`  
`fqualify( ./aa ) = /ACCT/GRP/aa --or-- /CWD/aa`
- `wordcnt(str[,delims][,start])` - counts tokens starting at str[start]  
`wordcnt( abc def,ghi ) = 3`
- `xword(str[,delims][,nth][,end_var][,start])` - see word function too  
`xword( abc def,ghi ) = def,ghi`

# JINFO function

**Syntax: JINFO ( [#]S|Jnnnn , item [,status] )**

- **59 unique items: Exists, CPUsec, IPAddr, JobQ, Command, JobUserAcctGroup, JobState, StreamedBy**
- **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 ; other jobs w/ same user. acct if jobsecurity is LOW; other jobs in same acct if AM cap; any job if SM or OP cap**
- 

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

JINFO items:

Account Inpri PassExempt\*\*

CIPin \* IntroDate Priority

Command\* IntroTime Private\*

ConnectMin IPAddr\* Quiet

ConnectSec JobUserAccount Restart

Copies JobUserAccountGroup StdinSPID

CPULimit JobName StdinSPstate

CPUMillisec JobNum StdlistDelete

CPUSec JobQ StdlistSPID

Deferred JobSecurity\*\* StdlistSPstate

DegradeMode\*\* JobState StreamedBy\*

Exists JobType StreamedByDate

FmtIntroDate JSMAINPin\* StreamedByLdev\*

FmtIntroTime LdevIn StreamedByTime

FmtPriority LdevOut User

FmtStreamedByDate LocAttr\* UserAccount

FmtStreamedByTime Numbered UserAccountGroup

Group Outclass RawIntroDate

HomeGroup\* Outpri RawIntroTime

\_\_\_\_\_ RawStreamedbyDate

\* security check \*\* need SM or OP RawStreamedbyTime

# JINFO - examples

- **if JINFO ( HPLASTJOB, EXISTS ) then # you know the job exists, at least right now!**
- **if JINFO ( S543 , IPADDR ) <> then # Session 543 is connected via the network**
- **if JINFO ( target\_job, FMTPRIORITY ) = DQ then # target\_job is currently in the DQ dispatcher queue**
- **setvar state JINFO (HPLASTJOB, STATE , status) while status = 0 and state = WAIT do setvar state JINFO (HPLASTJOB, STATE , status) endwhile**
- **if JOBCNT( @J , list) > 0 then while JINFO (word(list), EXISTS ) do... setvar list xword(list)**

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

The JINFO function retrieves information about the target job or session. The value of this information is valid for the moment in time when it is initially obtained. It is possible that the data is no longer accurate by the time the CI programmer references the item.

By default, the CI reports errors when using JINFO. That is, if the specified job no longer exists, or is in a state where the requested item cannot be returned, the CI writes an error message to

\$STDLIST and sets the CIERROR variable. It may be useful to suppress the CI s error handling for JINFO by passing the status variable. When an error occurs and status is supplied, the CI returns all JINFO errors in this variable: no message is written to \$STDLIST, the CIERROR variable is not set and -1 is functionally returned. If status is passed, it must be checked prior to referencing the retrieved item.

Many of the JINFO items for any job/session on the system are available to ordinary users. Some items are restricted:

- a job/session can see all items on itself .
- an ordinary user can see all items for jobs logged on with the same user.acct , as long as JOBSECURITY is LOW -- if JOBSECURITY is HIGH (default) the restricted items are not available.
- an AM cap user can see all items for all job/sessions in the same account.
- SM and OP users can see all items for all job/session on the system.
- SM or OP is required to see the special global items such as: degradedmode , passexempt and jobsecurity .
- Lockwords are removed from the command for all users.

# 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

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

The JOBCNT function returns the number of job/sessions that match the job\_spec , regardless of the state of the matching job/sessions. In other words, JOBCNT does not filter based on whether the job is waiting, scheduled, executing, etc. The function return is valid only for the moment it is returned, as a system s job/session count can continually fluctuate.

It is possible to retrieve the job/session IDs for the matching jobs by passing the joblist\_var parameter. This unquoted argument names an existing or new CI string variable. It will be set to a list of matching job/session IDs of the form: J|Snnn, followed by a space, followed by the next ID, etc. For example: S123 S445 J9 S567 J10 Since CI string variables currently cannot exceed 1024 characters, it is possible that the joblist\_var passed to JOBCNT cannot contain all of the matching job IDs. This situation is only detected by comparing the number of tokens in the joblist\_var against the function return. For example: setvar cnt JOBCNT( @ ,jlist) if cnt <>wordcnt( jlist) then ... # not all matching jobs in variable Assuming three digit job numbers, approximately 204 matches will fit in the joblist\_var variable. Possible solutions to this

restriction are:

- use separate JOBCNT calls for jobs and sessions
- use separate JOBCNT calls for various target accounts

There are no restrictions on the use of JOBCNT. Any user, regardless of their capabilities, can specify any job\_spec and retrieve the matching job/session IDs.

# JOB CNT - examples

- Find number of sessions logged on to SYS account: `calc JOB CNT( @S:@.SYS )`
- Display IDs for all waiting jobs: `if JOB CNT( @J , jobids) > 0 then setvar x 0 while setvar(job,word(jobids,,setvar(x,x+1))) <> and & JINFO( job, STATE ) = WAIT do echo #!job endwhile endif`
- Find all job/sessions logged on as you: `JOB CNT( @! hpjobtype:!hpjobname, & !hpuser.!hpaccount )`
- 
- 

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

Here is a more robust example to handle the `joblist_var` overflow condition:

```
setvar cnt JOB CNT(pattern, jlist)
```

```
echo !cnt job/session(s) match !pattern .
```

```
if cnt >0 then
```

```
if cnt <>SETVAR(wcnt,WORDCNT(jlist)) then
```

```
echo Some job/sessions matching !pattern could not be selected
```

```
echo Only the first !wcnt entries will be processed.
```

```
endif
```

```
setvar cnt 0
# process each selected job
while SETVAR(job,WORD(jlist,,SETVAR(cnt,cnt+1))) <>do
if JINFO(job, exists ) the
# job still exists for the moment...
echo Processing job ID #!job
# now do it
endif
endwhile
endif
```

# FINFO and FLABELINFO

## • New FINFO and FLABELINFO items: Item Mnemonic Size Description -----

- 58 sectors int16 number of sectors
- 59 extents int16 number of extents
- 60 createtime int32: for FLABELINFO (CLOCK fmt) strng: for FINFO- file creation time ( HH:MM AM / PM )
- -60 intcreatetime n/a file creation time (HHMMSS)
- 61 accessors int32 number of accessors
- 62\* long 64 bit file limit
- 63\* long 64 bit number of sectors
- 64\* int 1=file is large, 0 not large \* for Large file support coming in release 6.5

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

### Notes:

- 
- Large file support expected to be available in MPE release 6.5
  - Maximum large file will be 128 GB of data.
  - Design allows for larger files in the future.
  - CI variables still limited to signed 32 bits, thus there are no FINFO items that correspond to the new large file FLABELINFO items.

# New CI Variables (in 6.0 PP1)

- **HPDATETIME** - current date/time as **YearMonthDayHourMinuteSecondMillisecond**.
  - Ex. 19990816130510200 = Aug. 16, 1999 at 1:05:10.2 PM
  - Caution! when extracting date and time
- **HPDOY** - current day of the year (Jan. 1 = 1)
- **HPHHMMSSMMM** - current time in **HourMinuteSecondMillisecond** format.
  - Ex. 095421600 = 9:54:21.6 AM
  - Note: Milliseconds rounded to nearest tenth of a sec.
- **HPLEAPYEAR** - **TRUE** if current year is a leap year.
- **HPYYYYMMDD** - current date in **YearMonthDay**.
  - Ex. 19990911 = Sept. 11, 1999

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

- 
- **HPDATETIME** - CI variable containing the current date and time in a string formatted as "YearMonthDateHourMinuteSecondMillisecond". For consistency, if **HPDATETIME** is referenced more than once for a "logical" (or atomic) operation, such as extracting the date and the time, it is very important to store **HPDATETIME** into a user-defined variable and extract

from that variable. For example, to accurately separate the date and time from HPDATETIME the code below should NOT be used: `setvar mydate lft(hpdatetime,8) /* Don't do this! */ setvar mytime rht(hpdatetime,9) /* Don't do this! */` The problem above is that "MYDATE" could be set a millisecond before midnight, and "MYTIME" could be set a millisecond past midnight, with the result being: MYDATE = 19990314 /\* Mar 14 just before midnight \*/ MYTIME = 000000100 /\* Mar 15 just past midnight \*/ Instead, create a variable that contains HPDATETIME and extract from that variable. For example: `setvar date_time HPDATETIME /* This is correct! */ setvar mydate lft(date_time,8) setvar mytime rht(date_time,9)` Note: current time resolution is only tenths-of-a-second so the last two string characters will both be "0". Type: String, read-only. Example: if the current date and time is Feb 21, 1999 at 14:08:15.2, HPDATETIME equals "19990221140815200"

- HPDOY - A CI variable containing the day number in the current year, with Jan. 1 being day 1. Type: Integer, read-only. Example: On Feb, 7, 1999 HPDOY equals: 38
- HPHHMMSSMMM - A CI variable containing the current time in a string formatted as "HourMinuteSecondMillisecond". Note: current resolution is only tenths-of-a-second so the last 2 string characters will both be "0". Type: String, read-only. Example: if the current time is 2:08:15.2 PM, HPHHMMSSMMM equals "140815200"
- HPLEAPYEAR - A CI variable that indicates if the current year is a leap year. Type: Boolean, read-only. Example: On Mar. 14, 1999 HPLEAPYEAR equals: FALSE On Jan. 1, 2000 HPLEAPYEAR equals: TRUE
- HPYYYYMMDD - A CI variable containing the current date in a string formatted as "CenturyYear-of-centuryMonthDate". Type: String, read-only. Example: if the current date is Jul 14, 1999, HPYYYYMMDD equals "19990714"

## New CI Variables (cont.)

- **HPSPOOLID** - contains the \$STDLIST spoolfile ID (Onnnn) for the current job.
- **HPLASTSPID** - contains the \$STDLIST spoolfile ID for the job referenced in the HPLASTJOB variable.

- `:stream myjob :print !HPLASTSPID.out.hpspool`

- **HMSGFENCE** - controls CI errors vs. warnings and now IF THEN / ELSE messages.

**suppress false cmds (8) suppress warns only (1)**

**bits 0 26 27 28 29 30 31**

**suppress false cmds and \*\*\* msgs (16) suppress errors & warns (2)**

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

### Notes:

---

- HPLASTSPID and HPSPOOLID (5.5 PP3) contain a string of the form: Onnnnn . In a session HPSPOOLID is the empty string, .
- The HPLASTJOB variable can be modified so it is possible to see spoolfile Ids for any job. :

```
setvar hplastjob J432 :print !hplastspid.out.hpspool
```

- HPMSGFENCE (6.0) values:

bits 29..31 : 0 = show all errors and warnings

1 = show all errors but suppress warnings

2 = suppress all errors and warnings

bits 26..28 : 0 = show FALSE commands and \*\*\* expression true/false msgs

1 = shows \*\*\* expression true/false msgs, suppress FALSE cmds

2 = suppress \*\*\* msgs and FALSE commands

value setting -----

0 show: \*\*\* msgs, false cmds, warns, errors

1 show: \*\*\* msgs, false cmds, errors; suppress: warns

2 show: \*\*\* msgs, false cmds; suppress: warns, errors

8 show: \*\*\* msgs, warns, errors; suppress: false cmds

9 show: \*\*\* msgs, errors; suppress: false cmds, warns

10 show: \*\*\* msgs; suppress: false cmds, warns, errors

16 show: warns, errors; suppress: \*\*\* msgs, false cmds

17 show: errors; suppress: \*\*\* msgs, false cmds, warns

18 suppress: \*\*\* msgs, false cmds, warns, errors

# Date Intrinsic

- 
- **HPDATECONVERT, HPDATEFORMAT,**
- **HPDATEDIFF, HPDATEOFFSET,**
- **HPDATEVALIDATE,**
- **HPCALENDAR, HPFMTCALENDAR**
- **CI date variables (HPYYYY, etc) use HPCALENDAR now.**
  - see [jazz.external.hp.com](http://jazz.external.hp.com) and Communicator

[Previous slide](#)

[Next slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

---

I32V \* I32V \* I32 I32V

- HPDATECONVERT(inputcode,inputdate,outputcode,outputdate,status,cutoff)

I32V \* CA CA I32 I32 I32V

- HPDATEFORMAT(datecode,inputdate,formatspec,fmtdate,fmtdatelen,status,cutoff)

I32V \* \* I32 I32 I32V

- HPDATEDIFF(datecode, firstdate,seconddate,diffindays,status,cutoff)

I32V \* I32V \* I32 I32V

- HPDATEOFFSET(datecode,inputdate,offset,outputdate,status,cutoff)

I32 I32V \* I32V

- result := HPDATEVALIDATE(datecode,inputdate,cutoff)

- 

I32

- date := HPCALENDAR

I32V CA

- HPFMTCALENDAR(date,formatdate)

# POSIX Improvements

## • POSIX installation improvements:

- POSIX files have the correct file permissions
- more Unix utilities in their expected directory
- /etc/profile enhancements:
  - .bak version kept
  - /etc/profile.local executed at end
  - able to define shell vars from the CI
- the SH UDC in HPPXUDC.pub.sys is more intuitive
  - CWD no longer changed
  - e.g. sh -c makefile

## • Extended characters in POSIX filenames

- ~ ` \$ % ^ \* + { } | :

[Previous slide](#)

[Back to first slide](#)

[View graphic version](#)

## Notes:

- 
- JAVA (1.1.5), SAMBA, BIND/DNS in base 6.0 FOS. JAVA 1.1.7 in 6.0 PP1. APACHE and LDAP in 6.0 PP2. All supported by HP.
  - I0036431.HP36341.SUPPORT defines the permissions of many of the POSIX files. This file was modified to grant all user read access to all installable text files, execute access to initialization files such as /etc/profile, and traverse directory access (TD on MPE and X on unix) for all common directories.
  - 15 new symbolic links created, mostly for networking services: /bin/ftp, .bin/telnet, /etc/protocols, /etc/services, /etc/hosts, /etc/resolv.conf

- 3 device files have been established: /dev/null, /dev/tty and /dev/tape
  
- shell initialization: /etc/profile is executed for system wide initialization every time the shell is run. The file \$HOME/.profile is executed for user-specific initialization.
  - If /etc/profile.local exists it will be executed at the end of /etc/profile. This allows customizable system wide profiles to be maintained independently of new versions of the HP file /etc/profile.
  - The MANPATH and TERM shell variables defined in /etc/profile assume the value of a CI variable of the same name if one exists. The PATH shell variable has . removed and /usr/local/bin added. The HOME shell variable is defined as /ACCT/HOME\_GROUP. If the user does not have a home group their logon group is used instead. The HISTFILE shell variable is defined as \$HOME/.sh\_history. Since HISTFILE is now defined the SHUDC no longer needs to change your CWD to your home group.
  - The popular ll (ls -l) alias is defined.
  - The cursor is no longer homed followed by a clear-display when the shell is invoked.
  
- All existing archive libraries in /lib and /usr/lib are converted into shared libraries (.sl)