HP 3000 Computer Systems

MPE V INTRINSICS

Temporary Supplement

HEWLETT PACKARD
19447 PRUNERIDGE AVENUE, CUPERTINO, CA 95014

Part No. 32033-90007
E0784

Printed in U.S.A. 07/84

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

PREFACE

This is a temporary supplement to the MPE V Intrinsics Reference Manual. It is designed to cover all the intrinsics that are new or have been enhanced for MPE V releases E/F.00.00 and G.00.00.

While not designed as an update package, this temporary supplement should be used along with the MPE IV Intrinsics Reference Manual (30000-90010) to provide you a comprehensive reference source for MPE V intrinsics.

Currently the MPE V Intrinsics Reference Manual is undergoing a complete rewrite. In order to better serve you, we are trying to incorporate many of the comments and suggestions we have received via our Reader Comment Sheets and Service Requests. Unfortunately, the new Intrinsics Reference Manual is not ready for printing at the time of this software release. To meet your needs in the interim, we have compiled this temporary supplement to the MPE IV Intrinsics Reference Manual.

We hope that this temporary supplement does not inconvenience you. Rest assured that the new MPE V Intrinsics Reference Manual (32033-90007) will soon be on its way to your office. It will be distributed via normal distribution channels.

CONVENTIONS USED IN THIS MANUAL

NOTATION	DESCRIPTION
COMMAND	Commands are shown in CAPITAL LETTERS. The names mus contain no blanks and be delimited by a non-alphabetic character (usually a blank).
KEYWORDS	Literal keywords, which are entered optionally but exactly as specified, appear in CAPITAL LETTERS.
parameter	Required parameters, for which you must substitute a value, appear in ***bold italics***.
parameter	Optional parameters, for which you may substitute a value, appear in *standard italics*.
[ ]	An element inside brackets is optional. Several elements stacked inside a pair of brackets means the user may select any one or none of these elements. Example: [A] [B] user may select A or B or neither. When brackets are nested, parameters in inner brackets can only be specified if parameters in outer brackets or comma place-holders are specified. Example: [*parm1*[,*parm2*[,*parm3*]]] may be entered as: *parm1*,*parm2*,*parm3* or *parm1*,,*parm3* or ,,*parm3* , etc.
{ }	When several elements are stacked within braces the user ***must*** select one of these elements. Example: { A } { B } user must select A or B.
...	An ellipsis indicates that a previous bracketed element may be repeated, or that elements have been omitted.
user input	In examples of interactive dialog, user input is underlined. Example: NEW NAME? STICK2
superscript^{C}	Control characters are indicated by a superscript^{C}. Example: Y^{C}. (Press Y and the CNTL key simultaneously.)
⟨ ⟩	⟨ ⟩ indicates a terminal key. The legend appears inside.
<<COMMENT>>	Programmer's comments in listings appear within << >>.
** Comment **	Editor's comments appear in this form.

FCLOSE

INTRINSIC NUMBER 8

Closes a file.

SYNTAX

```
                IV              IV              IV
FCLOSE(filenum,disposition,seccode);
```

The FCLOSE intrinsic terminates access to a file. This intrinsic applies to files on all devices. FCLOSE deletes buffers and control blocks through which the user process accessed the file. It also deallocates the device on which the file resides and it may change the *disposition* of the file. If you do not issue FCLOSE calls for all files opened by your process, such calls are issued automatically by MPE when the process terminates. All magnetic tape files are left off line after such FCLOSE calls, to indicate to the System Operator that they may be removed.

The FCLOSE intrinsic can be used to maintain position when creating or reading a labeled tape file that is part of a volume set. If you close the file with a *disposition* code of 3, the tape does not rewind, but remains positioned at the next file. If you close the file with a *disposition* code of 2, the tape rewinds to the beginning of the file but is not unloaded. A subsequent request to open the file does not reposition if the sequence (*seq*) subparameter of *formmsg* in FOPEN specifies NEXT or default (1). A disposition code of 1 (rewind and unload) implies the close of an entire volume set.

If an unlabelled magnetic tape is closed with a *disposition* code of 0, 1, or 4, and the tape was written to while open, FCLOSE writes three EOFs at the end of the tape before performing a rewind or rewind/unload. This ensures that all tapes have an acceptable number of EOF marks at the end. The three EOFs are written only after the last FCLOSE to occur before the rewind, and only if the tape was written on.

For circular files, deletion of disc space beyond the end-of-file is not allowed.

PARAMETERS

filenum *integer by value (required)*
A word identifier supplying the file number of the file to be closed.

disposition *integer by value (required)*
Indicates the disposition of the file, significant only for files on disc and magnetic tape (*disposition* is ignored by the Foreign Disc Facility). This *disposition* can be overridden by a corresponding parameter in a :FILE command entered prior to program execution. The *disposition* options are defined by the bit fields (13:3) and (12:1) as follows:

(13:3) Domain Disposition

=000 No change. The *disposition* code remains as it was before the file was opened. Thus, if the file is new, it is deleted by FCLOSE ; otherwise, the file is assigned to the domain to which it belonged previously. An unlabeled tape file is rewound. If the file resides on a labeled tape, the tape is rewound and unloaded.

=001 Permanent file. If the file is a disc file, it is saved in the system file domain. A new or old temporary file on disc will have an entry created for it in the system file directory. Should a file of the same name already exist in the directory, an error code is returned and the file remains open. If the file is an old permanent file on disc, this domain disposition has no effect. Also, if the file is stored on magnetic tape, that tape is rewound and unloaded.

=010 Temporary job file (rewound). The file is retained in the user's temporary (job/session) file domain and can be requested by any process within the job/session. If the file is a disc file, the uniqueness of the file name is checked. Should a file of the same name already exist in the temporary file domain, an error code is returned and the file remains open. When a file resides on unlabeled magnetic tape, the tape is rewound. However, if the file resides on labeled magnetic tape, the tape is backspaced to the beginning of the presently opened file.

=011 Temporary job file (not rewound). This option has the same effect as domain disposition 010, except that tape files are not rewound. In the case of unlabeled magnetic tape, if this FCLOSE is the last done on the device (with no other FOPEN calls outstanding) the tape is rewound and unloaded. If the file resides on a labeled magnetic tape, the tape is positioned to the beginning of the next file on the tape.

=100 Released file. The file is deleted from the system.

(12:1) Disc Space Disposition (for fixed, undefined, and variable format files.

=0 Does not return any disc space allocated beyond the end-of-file indicator.

=1 Returns to the system any disc space allocated beyond the end-of-file indicator. The EOF becomes the file limit. No records may be added to the file beyond this new limit.

Bit (0:12) are reserved for MPE and should be set to zero.

When a file is opened by the FOPEN intrinsic, a file count (maintained by MPE for each file) is incremented by one. When the file is closed, the file count is decremented by one. If more than one FOPEN is in effect for a particular file, its disposition is saved but not affected by the FCLOSE call until the file count is decremented to zero. Then the effective (saved) *disposition* is the smallest nonzero *disposition* parameter specified among all

FCLOSE calls issued against the file. For example, the file XYZ is opened three successive times by a process. The first FCLOSE *disposition* is 1, the second FCLOSE *disposition* is %14, and the third (and last) FCLOSE *disposition* is %12. The final *disposition* on the file XYZ will be *disposition* 1 (permanent file and no return of disc space).

seccode *integer by value (required)*

Denotes the type of security initially applied to the file, this is significant only for new permanent files (*secode* is ignored by the Foreign Disc Facility). The options are:

0 Unrestricted access: the file can be accessed by any user, unless prohibited by current MPE provisions.

1 Private file creator security: the file can be accessed only by its creator.

CONDITION CODES

CCE	The file was closed successfully.
CCG	Not returned by this intrinsic.
CCL	The file was not closed, perhaps because an incorrect *filenum* was specified or because another file with the same name and *disposition* exists in the system. Any outstanding write I/O's which failed will also cause the FCLOSE to fail (such I/O's as buffered writes which are done in background). Additionally, an illegal *disposition* (5, 6, or 7) may have been specified. This can be detected by FCHECK returning an *error* of 49.

SPECIAL CONSIDERATIONS

Split-stack calls permitted.

ADDITIONAL DISCUSSION

Communicator 3000 Volume 2 Issue 1 (5955-1770)

For further information on magnetic tape files and associated functions, refer to the MPE File System Reference Manual (30000-90236).

FFILEINFO

Provides access to file information.

SYNTAX

```
   O-V          IV          IV          BA
FFILEINFO(filenum [,itemnum1,itemvalue1]
                  [,itemnum2,itemvalue2]
                  [,itemnum3,itemvalue3]
                  [,itemnum4,itemvalue4]
                  [,itemnum5,itemvalue5]);
```

Itemnum/itemvalue parameters must appear in pairs. Up to five items of information can be retrieved by specifying one or more *itemnum/itemvalue pairs*. FFILEINFO is designed to allow for future changes so that new file information can be defined and accessed.

PARAMETERS

filenum *integer by value (required)*
MPE file number returned by FOPEN.

itemnum *integer by value (optional)*
Cardinal number of the item desired; this specifies which item value is to be returned. (Refer to "ITEM #", Table 1.)

itemvalue *byte array (optional)*
Returns the value of the item specified by the corresponding *itemnum*; the data type of the item value depends on the item itself. (Refer to "ITEM", Table 1.)

CONDITION CODES

CCE	No error.
CCG	Not used.
CCL	Access or calling sequence error.

ADDITIONAL DISCUSSION

Communicator 3000 Volume 2 Issue 1 (5955-1770)

MPE File System Reference Manual (30000-90236)

Table 1. Item Values Returned by FFILEINFO

ITEM#	ITEM	TYPE		UNITS
1	Filename	BA	(see FGETINFO)	
2	Foptions	L	(see FGETINFO)	
3	Aoptions	L	(see FGETINFO)	
4	Record	I	(see FGETINFO)	words/bytes
5	Device type	I	(see FGETINFO)	
6	Logical device number	L	(see FGETINFO)	
7	Hdaddr	L	(see FGETINFO)	
8	File code	I	(see FGETINFO)	
9	Record pointer	D	(see FGETINFO)	
10	EOF	D	(see FGETINFO)	
11	File limit	D	(see FGETINFO)	records
12	Log count	D	(see FGETINFO)	records
13	Physcount	D	(see FGETINFO)	records
14	Block size	I	(see FGETINFO)	records/bytes
15	Extent size	L	(see FGETINFO)	sectors
16	Number of extents	I	(see FGETINFO)	
17	User labels	I	(see FGETINFO)	
18	Creator ID	BA	(see FGETINFO)	
19	Label address	D	(see FGETINFO)	
20	Blocking factor	I	(see FOPEN)	
21	Physical block size	I		words
22	Data block size	I		words
23	Offset to data in blocks	I		words
24	Offset of Active Record Table	I	(RIO files)	words
25	Size of Active Record Table within the block	I		words
26	Vol. ID(label tape)	BA	(see Label Tapes)	
27	Vol. set ID(label tape)	BA	(see Label Tapes)	
28	Expiration date(Julian)	I	(see Label Tapes)	
29	File sequence number	I	(see Label Tapes)	
30	Reel number	I	(see Label Tapes)	
31	Sequence type	I	(see Label Tapes)	
32	Creation date(Julian)	I	(see Label Tapes)	

Table 1. Item Values Returned by FFILEINFO (Continued)

ITEM#	ITEM	TYPE	UNITS
33	Label type	I	(see labeled tapes)
34	Current# of writers	I	(see IPC)
35	Current# of readers	I	(see IPC)
36	File Allocation Date	L	(CALENDER format)
37	File Allocation	D	(CLOCK format)
38	SPOOLFILE Device file number (#0 or #1 number. If device is not spooled, intrinsic returns zero)	L	(see File Code)
39	RESERVED		
40	Disc or diskette device status	D	
41	Device type	I	
42	Device subtype	I	
43	Environment file name	BA	
44	Last disc extent allocated	I	
45	Filename from labeled tape HDR1 record	BA	
46	Tape density	I	
47	DRT number	I	
48	UNIT number	I	
49	Software interrupt PLABEL	I	
50	Real device number of the file	I	
51	Virtual device number	I	
52	Last modification time (CLOCK format)	D	
53	Last modification date (CALENDAR format)	L	
54	File creation date (CALENDAR format)	L	
55	Last access date (CALENDAR format)	L	
56	# data blocks in a variable length file	I	
57	# of the user label written to the file	I	
58	Number of opens for output	I	
59	Number of opens for input	I	
60	Terminal type, defined as:	I	

0-file's associated device is not a terminal
1-standard hardwire or multi-point terminal
2-the terminal is connected via a phone-modem
3-DS psuedo terminal
4-X.25 Packet Switching Network PAD (Packet Assembler Disassembler) terminal

FINTEXIT

INTRINSIC NUMBER 23

Returns from the user's interrupt procedure.

SYNTAX

```
   O-V          LV
FINTEXIT(intstate);
```

The FINTEXIT intrinsic returns from the user's interrupt procedure. Software interrupts are set according to *intstate*. If *intstate* is omitted, FINTEXIT defaults to software interrupts enabled.

PARAMETERS

intstate — *logical by value (optional)*
A logical value indicating the state of software interrupts through bit (15:1) as follows:
=0 Leave software interrupts disabled.

=1 Enable software interrupts.
Default: (15:1)=1.

CONDITION CODES

The condition code remains unchanged.

FINTSTATE

INTRINSIC NUMBER 24

Enables/disables all software interrupts against the calling process.

SYNTAX

```
 L                          LV
oldstate:=FINTSTATE(intstate);
```

The software interrupt facility enables users to perform FREAD/FWRITE completion processing with their own interrupt procedure. An FREAD/FWRITE call is necessary to initiate the I/O request. Both of these intrinsics return to the user's process as soon as the request has been started. When the operation completes, the user's program is trapped (or "interrupted") and goes to a user chosen interrupt procedure. This performs whatever processing is necessary and then resumes the user's original program.

Soft interrupts are "armed" for a particular file by specifying the interrupt procedure's *plabel* in an FCONTROL call with a *controlcode* of 48. Calling "FCONTROL 48" with a parameter of 0 will disarm the software interrupt mechanism. The file is then accessed in the previous manner.

NOTE

MPE inhibits software interrupts just before entering an interrupt procedure. This is done to stop unwanted nesting of the interrupt procedures. Each interrupt procedure should call FINTEXIT (Refer to FINTEXIT in this section) to re-enable other interrupts just before it exits.

Software interrupts are normally automatically inhibited before a Y^C trap procedure. The trap procedure may elect to allow software interrupts, however, by calling the FINTSTATE intrinsic. The RESETCONTROL intrinsic will restore the process's interrupt state to its pre-Y^C value (unless the trap procedure issues an FINTSTATE call, in which case RESETCONTROL makes no change).

When the software interrupt is executed, Q-4 will contain the file number of the file that caused the interrupt.

It is necessary to issue a call to the IODONTWAIT intrinsic against the file in order to complete the request. When reading the *target* parameter is ignored in the FREAD call. The data is moved to the array specified by the *target* parameter of IODONTWAIT.

FUNCTIONAL RETURN

oldstate — *logical*
The old state (enabled or disabled) of software interrupts is returned by this procedure.

PARAMETERS

intstate *logical by value (required)*
A logical value enabling/disabling **software interrupts as through bit (15:1)** as follows:
=0 Disable software interrupts.

=1 Enable software interrupts.

CONDITION CODES

The condition code remains unchanged.

SPECIAL CONSIDERATIONS

An uncompleted FREAD/FWRITE request may be aborted by issuing an FCONTROL call with a *control-code* of 43 (abort NOWAIT I/O).

Limitations:

- Only message files allow soft interrupts.
- No more than one uncompleted FREAD/FWRITE may be outstanding for a particular file.
- The interrupt is held off while the user is executing within MPE, with the following exceptions: PAUSE and IOWAIT will allow the interrupt. The interrupt handler's return stack marker in this case will be set to reinvoke the intrinsic.
- May not be used with remote files.

GETDSEG

INTRINSIC NUMBER 130

Creates an extra data segment.

SYNTAX

```
                L       I       LV
GETDSEG(index,length,id);
```

The GETDSEG intrinsic creates or acquires an extra data segment. The number of extra data segments that can be requested, and the maximum size allowed these segments, are limited by parameters specified when the system is configured. When an extra data segment is created, the GETDSEG intrinsic returns a logical index number to the calling process. This index number is assigned by MPE and allows this process to reference the segment in later intrinsic calls. The GETDSEG intrinsic also is used to assign the segment the identity that either allows other processes in the job or session to share the segment, or that declares it private to the calling process. If the segment is sharable, other processes can obtain its logical index (through GETDSEG) and use this index to reference the segment. Thus, the logical index is a local name that identifies the segment throughout any process that obtained the index with the GETDSEG call. The logical index need not be the same value in all processes sharing the data segment. The identity, on the other hand, is a job-wide or session-wide name that permits any process to determine the logical index of the segment. If the intrinsic is called in user mode, then the data segment is initially filled with zeros. When GETDSEG is called in User Mode, all subsequent calls to intrinsics that use *index* must now be in User Mode. Likewise, when GETDSEG is called in Privileged Mode, all subsequent calls to intrinsics that use *index* must now be in Privileged Mode.

PARAMETERS

index *logical (required)*

A word to which the logical index of the data segment, assigned by MPE, is returned. When GETDSEG is called in User Mode, *index* is a logical index of the assigned data segment; if an error is found, *index* will be set to %2000-%2003. When GETDSEG is called in Privileged Mode, *index* is the actual entry segment entry number for the data segment that was assigned.

length *integer (required)*

The maximum size of the data segment requested, if the segment is not yet created, or the word to which the maximum size of the segment is returned, if the segment already exists.

id *logical by value (required)*

A word containing the identity that declares the data segment sharable between other processes in the job/session, or private to the calling process. For a sharable segment, *id* is specified as a nonzero value. If a data segment with the same id exists already, it is made available to the calling process. Otherwise, a new data segment, sharable within the job/session, is created with this *id*. For a private data segment, an *id* of zero must be specified.

CONDITION CODES

CCE	Request granted. A new segment was created.
CCG	Request granted. An extra data segment with this identity exists already.
CCL	Request denied. An illegal *length* was specified (*index* is set to %2000), or the process requested more than the maximum allowable number of data segments (*index* is set to %2001), sufficient storage was not available for the data segment (*index* is set to %2002) or a stack expansion necessary to satisfy the request could not be done because the stack was frozen (*index* set to %2003). (Note that a stack expansion is usually not necessary to get an extra data segment), or not enough room in job definition table to make an entry for the extra data segment (*index* set to %2004).

SPECIAL CONSIDERATIONS

Data Segment Management (DS) capability required.

ADDITIONAL DISCUSSION

Commuicator 3000 Volume 2 Issue 1 (5955-1770)

JOBINFO

INTRINSIC NUMBER 180

Provides access to job/session related information.

SYNTAX

```
          IV     D       LA          IV       LA         I
JOBINFO(jsind,JS#nnn,status  [,itemnum1,item1,errornum1]
                             [,itemnum2,item2,errornum2]
                             [,itemnum3,item3,errornum3]
                             [,itemnum4,item4,errornum4]
                             [,itemnum5,item5,errornum5]);
```

JOBINFO provides access to information related to any job/session that is current to the system. This intrinsic is expandable, and is written so that the addition of further functionality will be straight forward.

PARAMETERS

jsind *integer by value (required)*

One of the following integers indicating whether the *JS#nnn* denotes a session or job:

1 *JS#nnn* is a job.
2 *JS#nnn* is a session.

JS#nnn *double (required)*

A double value, 32 bits, identifying a job or session for which information will be retrieved.

status *logical array (required)*

A two word logical array to report the overall success/failure of the call. Only the first word contains significant information. The success/failure of the call is indicated by the following returns:

0 Successful call. All *errornums* equal zero.

1 Semi-successful call. One or more *errornum(s)* were returned with nonzero values.

2 Unsuccessful call. All *errornums* were returned with nonzero values.

3 Unsuccessful call. Syntax error in calling sequence.

4 Unsuccessful call. Unable to retrieve *JS#nnn*.

5 Process terminated. The process terminated during the start of retrieval.

itemnum *integer by value (optional)*

Cardinal number of the item desired. This specifies which item value is to be returned (Refer to "ITEM#" in Table 2).

item — *logical array (optional)*
Name of a reference parameter (whose data type corresponds to the data type for the desired information) to which the desired information is returned (Refer to "ITEM" in Table 2).

All of possible *itemnum* and *item* parameters are output parameters with one exception. Item number 1 can be used for an input and output parameter. Item number 1 is an input parameter only if the user is identifying a job or session by parsing a character string containing the *logon id*, *[jsname,]username.acctname*. Otherwise, it is an output parameter. The maximum number of characters returned is twenty-six. The returned string will be left justified and padded with blanks.

errornum — *integer (optional)*
A returned integer specifying the success or failure of the retrieval of each item. The returned values are:

0 Successful information retrieval.

1 Invalid *itemnum* (item number).

2 Desired information not pertinent to the given *JS#nnn* (e.g., user specifies a session number and wishes to know if a job had RESTART option).

3 User has insufficient capability to access this information.

4 The desired information is no longer available (e.g., returned when spoolfiles disappear).

SPECIAL CONSIDERATIONS

A user without System Manager (SM) or Account Manager (AM) capability can only retrieve information about the jobs/sessions logged on under the user name and account. A user with AM capability, but not SM capability will be restricted to access information concerning account sessions and jobs; a user with SM capability will be able to retrieve information concerning all sessions and jobs. The exception to the above security will be access to items which are normally available to the user, through MPE commands, who does not have any special capabilities.

CONDITION CODES

There are no condition codes on the traditional sense, but the *status* parameter can be thought of as a condition code.

ADDITIONAL DISCUSSION

Communicator 3000 Volume 2 Issue 1 (5955-1770)

JOBINFO provides the user with three options to identify a job or session which is on the system. The three options are the following:

- Taking the caller's job/session as default.
- Specifying a specific job/session number.
- Specifying a logon id.

To retrieve information about the job/session executing JOBINFO, the user must specify the appropriate *jsind* (1 for session, 2 for job) and a *jsnnn* of 0D (double-word zero). Further, the *itemnum* of the first optional triple must not be a one (1) as this invokes option number three (described below). For example, to retrieve the logon id for the caller, from a session, a JOBINFO call might look like this:

```
JOBINFO( 1, jsnnn, STATUS,,,,1,jsname,ERROR1 );
```

Where:

- *Jsnnn* must be 0D.
- *Jsind* is 1 since JOBINFO is executed from a session, and default to the caller's session for the logon id.
- *Jsname* must be a logical array of 13 words.
- The session's session number will be returned through *jsnnn*.
- The second triple contains the *itemnum* of 1. When using the default job or session, the first triple can not contain an *itemnum* equaling 1.

It is not possible to attempt a default call with *jsind* equaling 2 if JOBINFO is executed from a session. Likewise, it is not possible to call JOBINFO, attempting a default call, with *jsind* equaling 1 when JOBINFO is executed from a job.

The user may identify a job or session by specifying the appriopriate job/session number through *jsnnn*, along with the appropriate *jsind*. By supplying a non-zero *jsnnn*, the other two job/session identification options are over-ridden. Their is no restriction the use of any of the optional triples, or *itemnums*. An example of specifying a specific job/session number is a follows:

```
JOBINFO( 2, jsnnn, STATUS, 1, jsname, ERROR1 );
```

Where:

- *Jsind* equals 2 and specifies that *jsnnn* is a job number.
- *Jsnnn* is a job number.
- *Itemnum* equals 1 denoting that the logon id of job number *jsnnn* is to be retrieved.

The user may identify a job or session by specifying the appropriate job/session by supplying the appropriate logon id through the first optional triple. The user must supply the appropriate *jsind* and *jsnnn* must be 0D. The logon id is specified through the first triple with *itemnum* equal to 1, and *item* being a logical array containing the logon id (a character string). The logon id must be terminated by a binary zero (0). The maximum length of the logon id is twenty-six (26) characters, plus one (1) for the binary zero terminator. An example of this is as follows:

```
JOBINFO( 2, jsnnn, STATUS, 1, LOGON'ID, ERROR1 );
```

Where:

- *Jsind* equals 2, denoting the logon'id supplied is that of a job.
- *Jsnnn* equals 0D.
- The job number of the job denoted by LOGON'ID will be returned through *jsnnn*.
- The first triple is specified with an *itemnum* equal to 1, a logical array LOGON'ID containing the logon id of a job (terminated by a binary 0), and an integer error return for the item.

An example of initializing LOGON'ID might be as follows:

```
MOVE LOGON'ID(0) := "TESTJOB,LARRY.OSE",0;
```

Table 2. Item Descriptions

ITEM#	ITEM (information returned)	DATA TYPE
1	[JSNAME,]user.account (See note 1)	LA
2	session/job name (See note 2)	LA
3	user name (See note 2)	LA
4	user logon group (See note 2)	LA
5	user account (See note 2)	LA
6	user home group (See note 2)	LA
7	session/job introduction time (See note 3)	LA
8	session/job introduction date (See note 4)	LA
9	input ldev/class name (See note 2)	LA
10	output ldev/class name (See note 2)	LA
11	current job step (See note 5)	LA
12	current number of active jobs	I
13	current number of active sessions	I
14	job input priority	I
15	job/session number	D
16	jobfence	I
17	job output priority	I
18	number of copies	I
19	job limit (system)	I
20	session limit (system)	I
21	job deferred (See note 6)	I
22	main PIN - CI PIN for job/session	L
23	original job-spooled (See note 6)	L
24	RESTART option (See note 6)	L
25	sequenced - job (See note 6)	L
26	term code (See note 7)	L
27	CPU limit	L
28	session/job state (See note 8)	L
29	user's local attributes	L
30	$STDIN spoolfile number (See notes 9 & 10)	D
31	$STDIN spoolfile status (See notes 9 & 11)	I
32	$STDLIST spoolfile number (See notes 9 & 10)	I
33	$STDLIST spoolfile status (See notes 9 & 11)	I
34	length of current job step of item number 11	I
35	:SET $STDLIST=DELETE invoked (See note 12)	L
36	Job Information Table data segment number	L

Table 2. Item Descriptions (Notes)

1. Can be used as an input or output parameter. If used as an input parameter, a maximum of 26 ASCII characters, plus one for a binary 0 terminator is allowed. The input string must be in the form of [jsname,]user.account. The wildcard character @ is not allowed. If used as an output parameter, the logical array must be 13 words long. Output is left-justified and padded with blanks.

2. A ASCII output parameter. Logical arrays must be 4 words long output is left justified and padded with blanks.

3. Returns a 32-bit double word in a form to be used by the FMTCLOCK intrinsic.

4. Returns a 16-bit logical word in a form to be used by the FMTCALENDAR intrinsic.

5. Returns a maximum of 283 ASCII characters, and is the image of the command currently executing. The logical array must be long enough to accomodate the expected command image.

6. Returns the values: 0 - No
   1 - Yes

7. Returns the values: 0 - Regular terminal
   1 - Regular terminal with special log on.
   2 - APL terminal
   3 - APL terminal

8. Returns the values: 2 - Executing
   4 - Suspending
   32 - Wait
   48 - Initialization

9. Returns data for current jobs and sessions. $STDIN/$STDLIST files only.

10. Returns the spoolfile number as an integer.

11. Returns the values: 0 - Active
   1 - Ready
   2 - Open
   3 - Reserved

12. Returns the values: 0 - $STDLIST will be saved.
   1 - :SET $STDLIST=DELETE is invoked.

PROCINFO

INTRINSIC NUMBER 111

Provides access to process information.

SYNTAX

```
   O-V      I         I     IV        I         BA
PROCINFO(error1,error2,pin  [,itemnum1,item1]
                            [,itemnum2,item2]
                            [,itemnum3,item3]
                            [,itemnum4,item4]
                            [,itemnum5,item5]
                            [,itemnum6,item6]);
```

PARAMETERS

error1 *integer (required)*
An integer indicating the success or failure of the intrinsic call as described in Figure 1.

error2 *integer (required)*
An integer which supplies additional information concerning an error reported in *error1*. It is also defined in Figure 1.

pin *integer by value (required)*
An integer specifying the process identification number for which information is to be returned. A *pin* value of zero will return information about the calling process. Note that it is not compatible with the *pin* parameter of the GETPROCINFO intrinsic.

itemnum *integer (optional)*
An integer containing the item number (in any order) of an information option as defined in Figure 2. The user may request up to 6 options to be returned.

item *byte array (optional)*
Arrays (in the same order as the *itemnums*) of returned information as specified in Figure 2.

The parameters *error1*, *error2*, *pin* are required. The *itemnum* and *item* parameters are optional. The actual number included depends upon the information desired. The *itemnums* and the *items* are paired such that the *nth itemnum* corresponds to the *nth item*. An *itemnum* contains the option number of the desired information. The information is returned in the corresponding *item* or is stored using the *item* element as a pointer, depending on the information desired.

CONDITION CODES

CCE — Successful call. All error codes set to zero.

CCG — Not used.

CCL — Unsuccessful call with errorcodes set accordingly.

error1#	Meaning	error2
0	successful execution — no error	0
1	insufficient capability to return request information	index of offending itemnum
3	required parameter address (other than "error1") out of bounds	not used
4	address bounds violation while processing an option	index of offending itemnum
5	invalid item number	index of offending itemnum
6	invalid pin number no information returned	−1
7	unassigned pin number	−1
8	unpaired itemnum/item parameters	index of offending itemnum/item pair

Note 1: The process will abort if error1 parameter address is illegal or if the intrinsic is called in split stack mode.

Note 2: If an error condition is detected while processing an information request, the index of the itemnum where the offending option was located is stored in error2.

Figure 1. Error Codes Returned Form PROCINFO

item #	information returned	item
0	ignored	ignored
1	process identification number of calling process	integer where PIN will be returned
2	process identification number of the father of the specified process	integer where PIN will be returned
3	number of sons of the specified process (direct descendants)	integer where the number of sons will be returned
4	number of descendants (both direct and indirect) of the specified process	integer where the number of descendants will be returned
5	number of generations (number of levels in the process tree substructure) the specified process has including itself	integer where number of generations will be returned
6	process identification numbers of all sons (direct descendants)	integer array where son PINs will be returned (see note 1)
7	process identification numbers of all descendants (both direct and indirect)	integer array where descendent PINs will be returned (see note 1)
8	priority number in the master queue of specified process	integer where the priority will be returned (same as word 1 of the GETPROCINFO intrinsic)
9	state and activation information of the specified process	logical where the information will be returned (same as word 2 of the GETPROCINFO intrinsic)
10	program name where the specified process is currently executing	byte array where the fully qualified program name will be stored (see note 2)

Note 1: Some options return a variable number of PINs. In these cases ***item*** should be set by the calling process to point to an integer where the PINs will be returned. The first word of the array should be set by the calling process to indicate the array size in words. PINs will be stored into the array, one PIN per word, starting with the second word and continuing until the array is filled or all PINs have been returned. If the array is not filled, the remaining unused locations will be zeroed.

Note 2: The byte array for the program name must be a minimum of 28 bytes long. The name will be returned in the form of "f.g.a" where "f" will be the local file name, "g" will be the group name, and "a" will be the account name of the file containing the program that the specified process is currently executing. The name will be returned left-justified with the unused locations filled with blanks.

Note 3: If the calling process is executing in privileged mode, requests for information will be honored for any process. Otherwise, requests will be honored as follows:

1. Complete information will be returned for sons of the calling process itself.
2. Item ten will be returned only if the calling process has read access to the program file.
3. Information returned for indirect descendants and processes directly above the calling process will be limited to items two through seven and ten only.

Process handling capability will also be required for any user mode call unless the calling process is requesting information about itself.

Figure 2. Information Options for PROCINFO

Part No. 32033-90007
Printed in U.S.A. 7/84
E0784

HEWLETT PACKARD