TSX–Plus Version 6.40

Release Notes

These release notes describe the differences between TSX–Plus version 6.40 and version 6.31. The *TSX–Plus Documentation Set* incorporates the information described in earlier release notes. In case of any differences between the release notes and the manuals, the release notes take precedence.

January 16, 1989

Distributed and Supported by
Omnex CORPORATION 2483 Old Middlefield Way • Mountain View • California 94043
(415) 966-8400

The information in this document is subject to change without notice and should not be construed as a commitment by S&H Computer Systems, Inc. S&H assumes no responsibility for any errors that may appear in this document.

Contents

1 Support for Separate I– & D–Space

1.1 How to break the 64 Kb barrier

The PDP–11 architecture has a 16–bit address range, which implies that programs may only directly access at any single instant up to 64 Kb of memory. Several schemes have been developed to circumvent this program size limitation, such as disk and virtual overlays, virtual arrays, shared run-time regions, automatic program segmentation and program chaining. However, the architectural restriction of a 16–bit address space means that use of any of these techniques imposes some extra computational or I/O overhead. The PDP–11/23 has eight mapping registers for user programs which TSX–Plus uses to position jobs throughout physical memory. This means that each mapping register controls up to 8 Kb of user memory (64 Kb address space / 8 mapping registers = 8 Kb per register). The memory management hardware of more powerful PDP–11 implementations supports two separate sets of eight user mapping registers, one for instructions and one for data. This allows simultaneous mapping to 64 Kb of instructions (I–space) and separate mapping to an additional 64 Kb of data (D–space). The processors which have memory management units that support separate I– and D–space are: 11/73, 11/83, 11/44, 11/84, and the Pro–380. This feature is not available on the 11/23, 11/24, 11/34 or Pro–350.

Starting with TSX–Plus V6.40, it is now possible for programs which use certain special coding techniques to take advantage of the separate I– and D–space memory management hardware on processors which support it. This allows programs to have up to 64 Kb of program space and 64 Kb of data space, for a total instantaneous addressing capability of 128 Kb. Separate I– and D–space may be used in conjunction either with shared run-time regions or with PLAS regions. This makes the additional addressing capacity of separate I– and D–space most conveniently available when using language processors and run-time systems that transparently manage I– and D–space mapping. Currently, DISC plans to use this feature to improve performance in future releases of DBL for TSX–Plus.

Separate I– and D–space is expected to be used in combination with shared run-time code regions mapped through I–space or PLAS regions mapped through either I– or D–space. Special shared run-time fast mapping works as before. If separate I– and D–space is not enabled, then shared run-time mapping and fast mapping are unchanged from their behavior in previous versions.

For the adventuresome, the next sections present a brief description of program virtual memory organization and an overview of the memory management services available to the TSX–Plus programmer as background for the subsequent description of the system services which have been implemented to support separate I– and D–space addressing.

It is not within the scope of this discussion to present a tutorial on separate I– and D–space programming techniques. Briefly, instruction fetches are made from I–space and data references are to and from D–space. See the *PDP–11 Architecture Handbook* from *Digital Equipment Corporation* for more information on the technical aspects of separate I– and D–space.

1.2 User memory organization and mapping

When a program is started, TSX–Plus obtains memory mapping information for the job from block 0 of the .SAV image. Among other things, it checks the program high limit, whether it is overlayed, the initial stack pointer, the job status word, location 0, and location 56 (used by SETSIZ). Based on the program size and characteristics and how much memory is allowable for the job, the memory mapping is set up, the program is loaded and its execution begun. This contiguous section of physical memory into which the program is loaded is called the **static region.** In general, the static region is set up so that the program is loaded with virtual addresses from 000000 up to 157777 (PAR 0 through PAR 6). PAR 7 is normally used to map to a simulated copy of RMON fixed offsets, and is mapped from 160000 through the top of RMON (typically about 700. bytes). Virtual programs are those which do not require direct access to simulated RMON and so can have a static region which extends all the way through PAR 7. (In this case, the contents of fixed RMON offsets may still be obtained by the .GVAL request.) Another common alternative is to map PAR 7 to the hardware I/O page. All these initial mapping permutations occur solely and exclusively through user mode I–space.

Once started, programs can further influence their memory mapping through PLAS requests, or by mapping to shared run-time regions, or through real-time requests. Using the new split I– and D–space requests, programs may simultaneously access more than 64 Kb of memory when using shared run-time or PLAS regions. When a program issues a request to map to an extended region, the appropriate memory management register settings are made (and copied into the job's context region for use in the event of later remapping).

A job's initial memory image and mapping is determined when loading a program (.SAV) image. It may be affected during execution by the job size or mapping EMTs described in the previous paragraph. The system routine used to establish a job's memory mapping when first loaded, after a context switch, after the job sizing EMT (0,141), or after certain extended memory operations is called SETMAP. The SETMAP routine uses the following algorithm:

- Set up mapping registers for I–space through the program top (which may extend into PAR 7).
- If separate I– and D–space is enabled, set identical values in the D–space mapping registers.
- Disallow access to I–space (and D–space if enabled) virtual addresses above the program top. (Attempting to access disallowed virtual addresses reults in a trap to 4.)
- Set PAR 7 mapping to the I/O page (if requested) through I–space (and identically through D–space if enabled). This is *not* treated as extended memory.
- If the program is not a virtual job and has not mapped PAR 7 to the I/O page, map part of PAR 7 to a simulated copy of RMON through I–space (and D–space if enabled). Note that the simulated RMON does not extend through the full PAR and references above its top are disallowed. This is *not* treated as extended memory.
- Map extended memory regions which should be directed to PLAS, shared run-times or regions mapped to a physical address with the real-time EMT (17,140). Shared run-times and real-

time mapped regions may only be mapped through I–space. PLAS regions may be mapped through either I– or D–space.

1.3 Mapping to extended memory regions

The static region of a job is always mapped by a call to SETMAP. Mapping to extended memory regions may be done by a variety of requests: real-time EMT 17,140 to map to a physical address (I–space only); mapping (1,143; or fast mapping with the TRAP instruction) to a shared run-time region (both I–space only); and PLAS requests (.MAP, WS.MAP set during .CRAW or fast mapping with the TRAP instruction) to extended memory regions (both either I– or D–space). The following paragraphs discuss special mapping notes (idiosyncrasies if you will) for these methods.

Note that combinations of shared run-time, real-time, and PLAS mapping are possible but are strongly discouraged.

1.3.1 The static region

The SETSIZ program can be used to set the size of the static region to be allocated when a program is started. (See the appendix on SETSIZ in the *TSX–Plus Programmer's Reference Manual* for more information on job sizing.) This may also be done with the LINK /K:n switch or internally to the program as follows:

```
        .ASECT
. = 000056              ;Address modified by SETSIZ
        .WORD   28.     ;# of K-words for program == 56 Kb
```

If a program is restricted to a reduced memory allocation with SETSIZ or with the MEMORY command or from within the program by the TSX–Plus EMT to change a job's size (EMT 375 0,141), then the highest valid static region address will be correspondingly restricted. This is done by establishing the program top address and then calling SETMAP. This does not prevent a job from mapping virtual addresses above this limit to physical memory, to shared run-times or to PLAS regions.

1.3.2 Real-time mapping

When PAR 7 is mapped to the IO page or to simulated RMON (with real-time EMTs 5,140 and 6,140 respectively or as a consequence of the RUN/IOPAGE command or the program being installed with the IOPAGE attribute), addresses 160000–177777 are not treated as extended memory. Thus, the value returned by the PHYADD EMT (0,140) will be incorrect and will instead return the physical address that would correspond to the specified virtual address offset from the physical base of the job.

The MAPPHY EMT to map to a physical address (17,140) is treated as extended memory and will be correctly handled by PHYADD (I–space only). MAPPHY simply sets the appropriate I–space

PAR and PDR values; it does not invoke SETMAP. It does not support D–space. When MAPPHY is executed and the specified size is zero (0), then the specified extended PAR is released and SETMAP is called.

1.3.3 Shared run-times

The EMT to associate with a shared run-time does not immediately affect job mapping. However, when it is called with zero (0) as the pointer to the region name, then all shared run-times are released. In this second form all extended PAR and PDR values are cleared, all PLAS regions are remapped and SETMAP is called. Note that a side effect of this behavior is that any regions mapped via MAPPHY will be unmapped.

The EMT to map to a shared run-time region (1,143) simply sets the appropriate I–space PAR and PDR values; it does not invoke SETMAP. These are treated as extended memory – PHYADD (the I–space form) will return correct values. Shared run-times may not be mapped through D–space.

The EMT to establish fast-mapping to shared run-time regions does not immediately affect job mapping but rather creates entries in the job's fast-map tables. See the section on fast-mapping below.

1.3.4 PLAS regions

PLAS regions may be mapped either through I–space or D–space and are always treated as extended memory regions. Thus, PHYADD will return correct results, but the correct form must be used (I–space or D–space).

When a PLAS region is mapped with the .MAP request or because the WS.MAP bit was set during a .CRAW request, the appropriate I–space or D–space PARs and PDRs are set immediately; SETMAP is not called.

When a window is unmapped with the .UNMAP request or as a consequence of the .ELAW or .ELRG requests, then the PAR and PDR registers affected by the window are cleared, as well as any affected fast-map entries. Then SETMAP is called.

The EMT to establish fast-mapping to PLAS windows does not immediately affect job mapping but rather creates entries in the job's fast-map tables. See the section on fast-mapping below.

1.3.5 Fast mapping

Fast-mapping can be used to map shared run-time regions into the job's virtual I–space addresses, or to map PLAS regions into the job's virtual I– or D–space addresses. Fast-map regions are treated as extended memory and PHYADD will return valid results. When the TRAP instruction is used to fast-map an extended memory region (either shared run-time or PLAS), then the appropriate PAR and PDR values are set immediately based on information in the job's fast-map tables. SETMAP is not called.

1.4 Enabling separate I– and D–space

System service calls (EMTs) have been defined to enable and disable separate I– and D–space user memory mapping. These EMTs do not by themselves increase a job's addressing capability, but may be used in conjunction with remapping a portion of the job's virtual address space to shared run-time regions or to PLAS regions. They are only supported on processors whose memory management hardware supports separate I– and D–space mapping and will return an error if not supported by the hardware.

When a program is loaded, user memory mapping for the job is initialized from virtual address 000000 through the program's memory allocation limit. This is the static region, mapped solely through I–space (by default), and both instruction fetches and data references are made to the static region through I–space. After execution of the EMT to enable separate I– and D–space, then the same static region is doubly mapped both through I–space and through D–space. Thus, although mapped separately, both address spaces refer to the same physical memory locations and the program behaves as though no change had been made to its mapping. However, at this point, portions of the I–space or the D–space may be separately remapped to different physical memory locations, thus doubling the total amount of physical memory which may be directly addressed by the program.

The form of the EMT to enable separate I– and D–space is:

```
EMT     375
```

with R0 pointing to an argument block of the form:

```
.BYTE   4,143
```

After successful execution of this request, the job's instruction and data space are mapped separately and may be manipulated (more or less) independently of each other. This separate mapping remains in effect until the program exits (or aborts) or until the following EMT is issued to disable separate I– and D–space mapping:

```
EMT     375
```

with R0 pointing to an argument block of the form:

```
.BYTE   5,143
```

These EMTs report the following error code:

Error Code	*Meaning*
6	The memory management hardware does not support separate I– and D–space.

1.5 I/O to separate I– and D–space regions

When I/O buffers specified in read or write requests reside in virtual addresses which are covered by separately mapped I– and D–space regions, then the data transfer is always directed to the D–space. (When separate I– and D–space is **not** enabled, then data transfers are directed to I–space.) The real-time EMT to convert a virtual address to a physical address (0,140) applies only to I–space. A new EMT (22,140) has been created to convert D–space virtual addresses to a physical address. When it is necessary to convert a virtual address to a physical address, be sure to use the appropriate request. If it is necessary to perform I/O to a buffer in an I–space region, then D–space must be first disabled.

The entire I/O buffers specified by read or write requests must be contiguously mapped. It is not allowed to specify an I/O transfer buffer which spans mapping boundaries. For example, assume a program has enabled separate I– and D–space and used PLAS mapping requests so that its memory organization looks something like:

	Virtual addresses	I–space	D–space
PAR 0	000000 – 017777	static	static
PAR 1	020000 – 037777	static	PLAS-1
PAR 2	040000 – 057777	static	PLAS-3
PAR 3	060000 – 077777	SRT-1	static
PAR 4	100000 – 117777	SRT-1	static
PAR 5	120000 – 137777	PLAS-2	static
PAR 6	140000 – 157777	PLAS-2	PLAS-3
PAR 7	160000 – 177777	RMON	RMON

In this example, portions of the job are mapped to six different contiguous segments of memory. Under TSX–Plus, the static region is always contiguous in physical memory, as are PLAS regions and shared run-time regions, although mapping windows into them need not be contiguous. Because I/O is never allowed to I–space regions, it would not be possible to read or write to either the SRT-1 shared run-time region or to the window into the PLAS-2 region. Although it would be allowed to write the simulated RMON region to disk, that would not be particularly useful, and it would be a *terrible* idea to read into it. So, I/O to the simulated RMON addresses is irrelevant. Transfers could be made to or from either of the PLAS-1 or PLAS-3 regions. However, no single transfer should be specified which would span the boundary between them, or between either of them and the static region.

TSX–Plus does not monitor or enforce restrictions on I/O requests which attempt to cross mapping boundaries — that responsibility remains with the programmer. Attempts to perform I/O across mapping boundaries will either result in unwanted results or will cause an I/O error for attempting to access unauthorized memory. For example, it might be possible to have mapped the PAR 2 and PAR 6 D–space virtual address windows to physically adjacent segments of the PLAS-3 region. Then, specifying a 12 Kb I/O request which started at 040000 (base of PAR 2) should go midway through PAR 3. However,

because of the way this job is mapped and the invalid crossing of a mapping boundary, the actual transfer would probably actually cover virtual addresses 040000 to 057777 and 140000 to 147777. Besides being stylistically offensive and a nightmare to debug, I/O requests which attempt to cross mapping boundaries are not supported by TSX–Plus and any unpredictable results which may result from them are explicitly disclaimed.

1.6 Using I– and D–space with Shared Run-Time Regions

When a program is loaded into memory its mapping registers are set up so that all memory references (both instruction and data) are mapped through I–space. Memory is allocated and mapped from virtual address 000000 up to the high limit of the program. This is called the static region. Then, when a program subsequently maps to a shared run-time region, portions of the the I–space mapping are altered to point into the shared run-time as specified. However, although the entire static program memory is still allocated to the job, portions of it are no longer accessible because they have been "mapped away". Now, with support for separate I– and D–space mapping, these unmapped portions of the static region can be used for data, mapped through D–space, while the same virtual address range is mapped through I–space to the shared run-time. (Note that mapping the shared run-time through D–space, leaving the static region mapped through I–space, is not allowed.)

The normal sequence of events for a program to use the unmapped portions of the static region as data space in conjunction with I–space mapping to a shared run-time region is as follows:

1. When a program is first loaded, only I–space is enabled, and all instruction fetches and all data references are made through I–space.
2. Enable separate I– and D–space (using EMT 375, 4,143). Both I– and D–space are now mapped identically to the same static region.
3. Associate with a shared run-time. This does not change the mapping.
4. Map a portion of the shared run-time into the program space. At this point, the virtual address range mapped to the shared run-time is split so that instructions will be fetched from the shared run-time and data references and I/O are directed to the remaining static region.
5. (... Do the real work ...)
6. Disassociate from the shared run-time. This restores mapping so that both I– and D–space are again identically mapped to the static region.
7. Disable separate I– and D–space (using EMT 375, 5,143). Now, the static region is again mapped only through I–space and all instruction fetches and data references occur through I–space.

1.7 Using I– and D–space with PLAS Regions

PLAS is an acronym for Program's Logical Address Space. This is a feature of the RT–11 XM (eXtended Memory) monitor which allows programs to address extended memory regions. TSX–Plus implements EMT's which are compatible with the PLAS requests of RT–11 XM so that programs which use PLAS features can be run without modification. The PLAS EMT's are used to implement FORTRAN virtual arrays (the FORTRAN IV OTSGEN VIRP option) and to support the virtual overlay handler from LINK. They may also be manipulated directly from MACRO programs.

When a PLAS region is created, an appropriate size region of memory is reserved from the pool of physical memory available to user jobs. If the region is "global", then it remains reserved until the region is eliminated and is never swapped out of memory. If the region is "local", then it may be swapped out to the PLAS region swap file if it becomes necessary to swap the job's static region. Creation of a PLAS region does not, by itself, affect the job's memory mapping. This is accomplished either explicitly by the .MAP request or implicitly as an option to the .CRAW request. PLAS regions may also be mapped using TSX–Plus "fast mapping" as described in section 2. PLAS regions are mapped by default through I–space, analogous to shared run-time mapping. However, TSX-Plus defines a bit in the Region Status Word which specifies that when separate I– and D–space has been enabled then all windows into the region should be mapped instead through D–space. Using separate I– and D–space mapping allows PLAS regions to be mapped either through I–space or through D–space, leaving the static regions (which otherwise would have been "mapped away") accessible through the other mapping space (D or I). In fact, it is possible to map some PLAS regions through I–space and simultaneously map other PLAS regions through D–space. TSX–Plus also defines a bit in the window status word which allows PLAS windows to have overlapping virtual address ranges, thus allowing complete remapping of a job's virtual address space — even to "map away" the static region when separate I– and D–space is enabled.

Note that when separate I– and D–space is enabled, the space through which windows may be mapped into PLAS regions is determined by region characteristics, not the window characteristics. This means that it is not allowed to map one window into a PLAS region through I–space and another window into the same region through D–space. The space through which all windows into a single region are mapped is determined by the state of the "use D–space" bit in the region status word at the time of region creation.

The following non-standard PLAS status bits are defined:

Word	New bit	Mask	Meaning when set
R.GSTS	RS.PVT	000001	This is a private region (defined since v4.0)
R.GSTS	RS.DSP	000002	This region is to be mapped through D–space (new)
W.NSTS	WS.OVR	002000	This window may overlap other windows (new)

The following new error codes are defined for the .CRRG request:

Error Code	*Meaning*
20	Attempt to create D–space region and hardware does not support separate I– and D–space.
21	Attempt to create D–space region and job has not already enabled separate I– and D–space mapping.

2 Fast mapping to PLAS regions — 20 times faster

Fast mapping using the TRAP instruction which was previously restricted to use with shared run-time regions now also supports global PLAS regions. (It may not be used with swappable PLAS regions — either unnamed or private named regions.) When compared with the old method used to map around within a PLAS region (.CRAW with WS.MAP), fast mapping within the PLAS region is about 20 times faster. Fast mapping also tracks whether the region is to be mapped through I–space (the normal case) or through D–space if the region was set up for D–space mapping. Fast mapping may be used in place of the .MAP request. Like the .MAP request, it uses information from the region and window control blocks to define the region to be mapped. The following request does not actually perform the mapping, but instead defines the fast mapping values to be used when mapping is requested by the TRAP 1 instruction. The form of the request is:

```
EMT     375
```

with R0 pointing to an argument block of the following form:

```
.BYTE   6,143
.BYTE   window-id, fast-map-number
.WORD   region-id
```

The *window-id* is the value returned by the .CRAW request, and is in the range 1–8. The *fast-map-number* is the value to be placed in R0 when the TRAP 1 instruction is executed to identify the fast-map region number, and is in the range 0–39. The *region-id* is the value returned by the .CRRG request. The PLAS region need not be currently mapped in order to associate a fast-map region with it. On successful return, R0 contains the actual length (number of 64-byte blocks) that will be mapped through the window by the TRAP 1 instruction — analogous to WS.LEN which is not maintained when using this mapping method.

See the descriptions of fast mapping to shared run-time regions in TSX–Plus V6.3 for more details on the use of fast mapping.

Errors returned by this request are:

Error Code	*Meaning*
1	Specified window or region is not defined
2	Invalid fast-map-number (must be 0-39.)
3	Invalid window-id (must be 1-8.)
4	Attempt to fast map to swappable region

3 Fast mapping extended to multiple PARs

Formerly, when a fast map region was defined, it was implicitly assumed that the region was 8 Kb or less. Now, it is possible to automatically define several fast-mapping regions and map to them in single operations. When the fast-map region size is defined to exceed 8 Kb, subsequent fast-map regions are also automatically assigned. If the combination of fast-map region number and size of the region would cause subsequent fast-map numbers to overflow the allowed limit (40 regions, numbered 0–39.), then an invalid fast-map region number error code will be returned. In addition, no checking is performed to prevent or warn about overwriting of previously defined fast-map regions. It is up to the user to manage fast-map region number assignments.

When a TRAP instruction is issued to cause fast-mapping to occur, any subsequent fast-map regions which are required to fully map the region are also automatically mapped.

4 Debugger I and D support

New functionality has been added to the TSX–Plus program debugger in order to support separate I– and D–space. (Remember that on processors which do not support separate I– and D–space or if separate I– and D–space is not currently enabled, then I–space and D–space are equivalent.) When instructions are decoded (e.g. with the "[" and "]" commands or when a breakpoint is hit), they always refer to I–space. When data values are examined (with the "/" and "\" commands) they refer by default to D–space. If separate I– and D–space is enabled, then these data reference commands can be forced to examine values in I–space mapping with the ";1I" command. Subsequent data references are directed to I–space until this mode is disabled with the ";I" (or ";0I") command. If separate I– and D–space is enabled, then data watchpoints always refer to D–space.

5 Flag page for spooled devices

As part of the spooling system, it is now possible to print a flag page at the beginning of each file sent to a spooled device. Flag pages are useful to readily identify the start of each new listing and help route it to the correct owner. Each flag page contains: the site name; the name, PPN, and number of the job which printed the file; the name of the file; and the time, day and date when the

file was sent to the spooling system. The flag page information may be centered for either narrow or wide forms (80 or 132 columns).

Flag pages are enabled separately for each spooled device, on a *per device* basis, not individually for each file. The initial selection of flag pages may be controlled by the SPFLAG macro in TSGEN. The SPFLAG macro accepts one parameter for each device specified in the SPOOL macro to enable ("F") or disable ("N") flag pages for the respective devices. A similar macro (SPWIDE) is used to identify whether the flag page for the respective device should be centered for narrow ("N"; 80 columns) or wide ("W"; 132 columns) forms. For example, the following combination of macros would initially select wide flag pages for LP and no flag pages for CL2 (but with narrow centering if they are enabled later).

```
SPFLAG  F,N     ;Flagpage on device 1, Noflagpage on device 2

SPWIDE  W,N     ;Wide pages on device 1, Narrow pages on device 2

SPOOL  2,20.,3,2000.,<LP CL2>,0,5.
```

The following new keyboard commands have also been added to the SPOOL command to dynamically control flag pages. (See section 10.2 for information on how to control flag pages from within programs.)

```
SPOOL ddn,[NO]FLAGPAGE

SPOOL ddn,{NARROW | WIDE}
```

where *ddn* is the name of the spooled device. FLAGPAGE enables flag pages to be printed at the beginning of each file subsequently sent to that spooled device and NOFLAGPAGE disables subsequent flag pages. NARROW enables narrow (80 column) flag page centering, and WIDE enables wide (132 column) flag page centering. Note that flag pages are generated at the time the first data is written to the spooled device. If a file has already been opened and the first data written to a spooled device, then subsequent commands to control flag pages for that device will have no effect on the flag page for that file – it will already have been generated or not according to the settings in effect at the time of the first write for that file.

6 Enhancements to the [NO]ACCESS Command

Two new features have been added to improve the ACCESS command. A NOACCESS command has been added, which works in conjunction with ACCESS restrictions to deny access to the specified devices. Also, it is now possible to specify a wild-card for the device unit number. The intention of these two changes is to increase the flexibility of the access restriction scheme within the limited space reserved for access entries.

When using a wild-card for the device specification, either the entire device may be wild-carded, or now just the device unit number may be wild-carded. For example, **ACCESS *:*.SAV/READ** would grant access to all executable programs (with .SAV extensions) on all devices. However, **ACCESS DL*:*.SAV/READ** would grant access to all executable programs (again with .SAV extensions) only on devices serviced by the DL handler, but those on other devices (such as DM or DU) would not be. This may be especially useful for CL lines, e.g. **ACCESS CL*:** instead of **ACCESS CL0:,CL1:,CL2:,CL3:,CL4:**

To summarize [NO]ACCESS wild-card handling, four parts of the specification may now be wild-carded: the device specification, the device unit number, the file name, or the extension. The following are valid wild-card specifications:

ACCESS Specification	Interpretation
*:A.DSK	A.DSK is accessible on any device
DU*:A.DSK	A.DSK is accessible on any DU unit
SY:*.OK	All .OK files on SY: are accessible
DL1:MARY.*	All MARY files on DL1: are accessible

The NOACCESS command is used to specify devices and files which may not be accessed. NOACCESS restrictions are only applied *after* ACCESS has been granted. Full access is tentatively granted to all devices and files if no ACCESS commands have been issued. That is, absence of any ACCESS commands is equivalent to an **ACCESS *.*** command. This allows the NOACCESS command to be used in those cases where it is only necessary to restrict access to a few devices or files.

The NOACCESS command accepts one switch, /WRITE, which is used to restrict write access to the file. If no switch is specified to the NOACCESS command, then no access is permitted to the file, either read or write. If the /WRITE switch is appended to the file specification, then read access is granted to the file, but it may not be written to.

It is important to remember that NOACCESS restrictions are applied only *after* ACCESS restrictions. If access to a file is not granted by either having no ACCESS commands or by matching an ACCESS specification, then access to the file is denied. If access to a file is tentatively granted either by having no ACCESS commands or by matching an ACCESS specification, then the NOACCESS table is checked and may be used to further restrict access to the file.

Besides [NO]ACCESS restrictions, there are a few other rules affecting access to devices and files. Most importantly, if the job has BYPASS privilege, then the are no access restrictions – the job always has full access to everything. Read and write access are always granted to TT:. Access to .SYS and .TSX files on the boot device (SY:) require SYSPRV privilege. Non-file structured requests require the corresponding NFSREAD and NFSWRITE privileges. After these tests, then ACCESS and NOACCESS restrictions are applied.

The following examples show some combinations of ACCESS and NOACCESS commands and their behavior with respect to certain file specifications.

This is an example of a traditional enumerated ACCESS list. Give read access to utility programs on the system disk and read-only access to all files in a shared logical subset disk also on the system disk. Give the individual full access to a work area, one floppy drive and three printers.

```
ACCESS SY:*.SAV/READ,DL0:COMMON.DSK/READ,DL1:RAYS.DSK
ACCESS DY0:,CL0:,CL1:,LP:

PRINT SY:STARTS.COM          Denied   – only .SAV files are allowed
COPY STARTS.COM SY:          Denied   – no write access to SY:
MOUNT LD0 SY:TEST            Denied   – only SY:COMMON.DSK is mountable
MOUNT LD1 DL0:COMMON         Allowed  – DL0:COMMON.DSK is read-only
COPY REPORT.DAT LD1:         Denied   – DL0:COMMON.DSK is read-only
MOUNT LD DL1:HENRYS          Denied   – only DL1:RAYS.DSK is mountable
MOUNT LD DL1:RAYS            Allowed  – DL1:RAYS.DSK is read/write
COPY SY:EDIT.SAV LD:         Denied   – Exception – PIP requires SY:/READ
INIT DY1:                    Denied   – DY0: is allowed, not DY1:
```

The intention in the following example is to give access to utility programs on SY:, deny access to one logical subset disk on SY:, and otherwise grant full access. However, because there was an ACCESS command, which only allows read access to .SAV images on SY:, nothing else is accessible. The NOACCESS command in this case does nothing.

```
ACCESS SY:*.SAV/READ
NOACCESS SY:PRIVAT.DSK

TYPE SY:STARTS.COM           Denied   – only SY:*.SAV files are readable
PRINT DL1:MISC.DAT           Denied   – only access is to SY:*.SAV
COPY TT:A TT:B               Allowed  – TT: access is always allowed
R KED                        Allowed  – SY:*.SAV files are readable
MOUNT LD DL1:COMMON          Denied   – only access is to SY:*.SAV
```

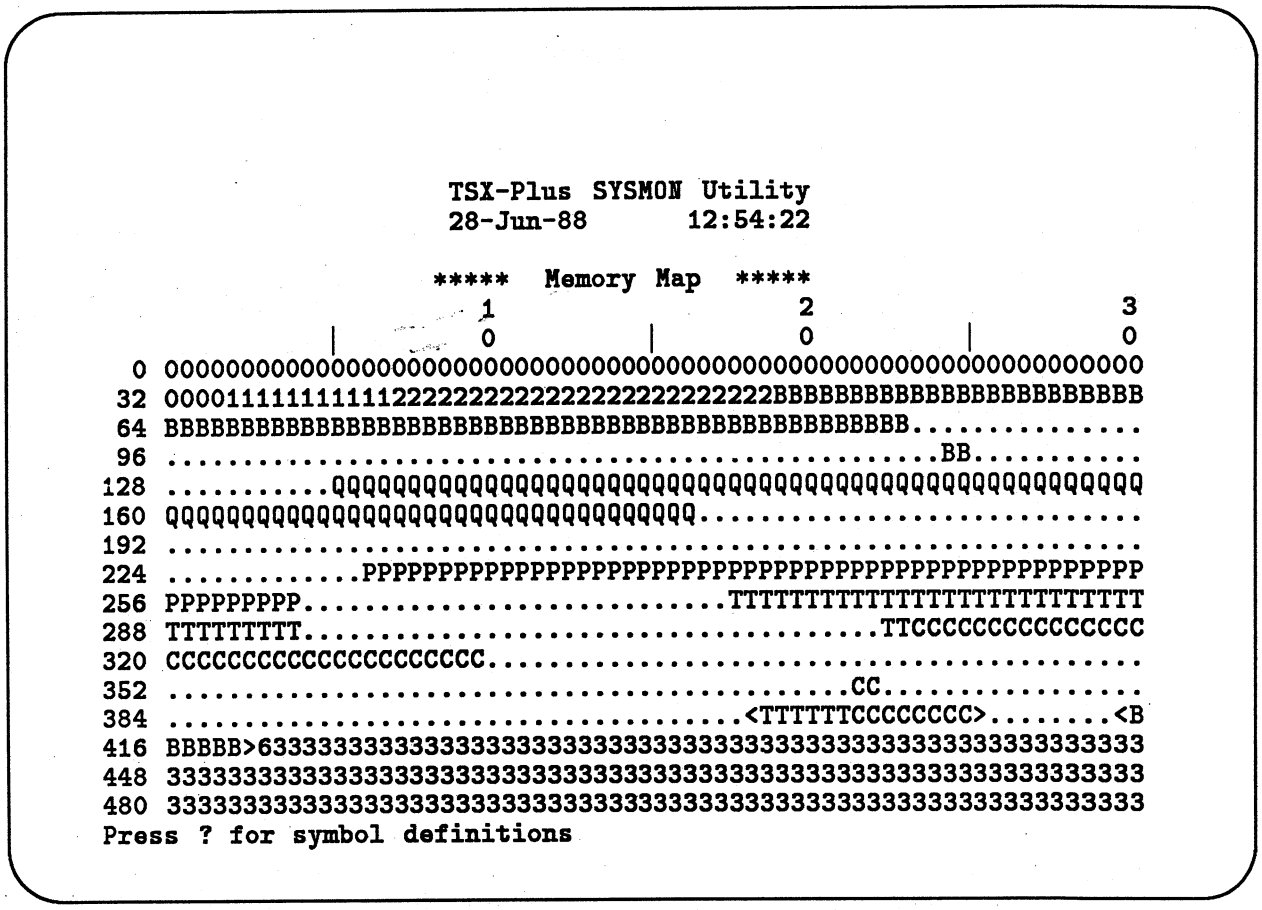
The next example was intended to grant access to all disks, except to restrict writes to the system disk. However, the limited read access is overridden by the full access to all DL units.

```
ACCESS DL0:/READ,DL*:

TYPE SY:STARTS.COM          Allowed  – full access to all DL units
PRINT SY:STARTS.COM         Denied   – no access to LP:
COPY STARTS.COM DL0:        Allowed  – full access to all DL units
```

This example shows the correct way to handle the above situation. Writes to the system disk are forbidden, but since there are no ACCESS commands, all other devices and files are fully accessible.

```
NOACCESS DU0:/WRITE

PRINT DU0:STARTS.COM        Allowed  – does not try to write to DU0:
COPY STARTS.COM DU0:        Denied   – tries to write to DU0:
COPY DU1:TEST.DAT DU2:      Allowed  – does not try to write to DU0:
MOUNT LD DU3:MYDISK         Allowed  – does not try to write to DU0:
INIT/NOQ LD:                Allowed  – DU3:MYDISK is writeable
```

7 New SYSMON memory map display



The SYSMON system monitoring utility has been enhanced by the addition of a memory map display. SYSMON can now be used to determine where different system regions as well as different user jobs are located in memory.

Each cell in the memory map represents a 512 byte page of memory. There are a maximum of 8 different screens with each screen representing the contents of 512 Kb of memory. The symbols for the different parts of memory are explained in the following help screen:

```
                    TSX-Plus SYSMON Utility
                    12-Jan-89       10:12:25

                 Memory Map Symbol Definitions

0 - TSX-Plus System Regions              A - Job  1 . . . Z - Job 26
1 - Device Handler Regions               a - Job 27 . . . n - Job 40
2 - Terminal I/O Buffers                 < - Beginning of a PLAS Region
3 - Cache Buffers                        > - End of a PLAS Region
4 - Mapped I/O Buffers                   . - Unused Portion of user Memory
5 - Performance Monitor Buffer           # - Memory above TSX-Plus or
6 - Memory Map Table                           Restricted Memory
7 - Shared Run-Time Regions
9 - VM Pseudo Disk Data Area

Press a number followed by a '^A' to display only that screen.
Press a number followed by a '^R' to display screens 0 through that number.
Press a symbol followed by a '^H' for a brief description of that symbol.

Examples:  0^A - displays only the first screen (   0 -  511 Kb)
           2^A - displays only the third screen (1024 - 1535 Kb)
           3^R - displays screens 0 through 3   (   0 - 2047 Kb)
 Press '?' to return to the Memory Map display
```

The memory display also has a mini-help facility. A one line description of any symbol in the memory display may be obtained by typing that symbol followed by a control-H. If that symbol is a letter representing a valid job number, then the line number, the program the job is running, the job name, and the job state are displayed. Note that case is significant in job symbols, i.e. 'b' $\neq$ 'B'.

8 Job status information

A special control character may now be used to obtain the current job status. The control character is normally defined as control–T, but may be changed by altering TSGEN parameter STATCH. This feature may be disabled by setting STATCH to zero. The displayed information contains: job number, job name, time, program name, job size, job state, and accumulated CPU time. For example:

```
5  Harry  16:27:03  KMON  38Kb  Wait-TI  CPU=0:00:01.9
```

In order to allow this facility to work without forcing an outswapped job back into memory, the display is generated in a manner similar to the SEND command. Consequently, the output is not handled by the Process Windowing screen manager. This has the beneficial side effect that the screen contents may be cleaned up by refreshing the window, although the control–T information cannot be captured by the printwindow facility.

The state descriptions are expanded more fully than in the SHOW JOBS command, but in order to keep the information on a single line, they are not as complete as those in the SYSMON Process execution status screen (screen 3). If the job is not currently in memory, "-Swap" is appended to the state description. The states are:

Executable states

State	Description
Real-time	Job priority at or above PRIHI
TT_sngl_done	Single character activation done
TT_input_done	Normal activation character received
TT_output_empty	All pending terminal output done
Interact_comp	Interactive class compute bound
Timer_done	.TWAIT or .MRKT expiration
TT_output_low	Terminal output nearly done
I/O_complete	I/O operation just completed
Compute	Normal compute bound
Low_prio_comp	Low–priority compute bound

Non-executable states

State	Description
Wait-IOQ	Waiting for free I/O queue element
Wait-MIO	Waiting to do mapped I/O
Wait-CSH	Waiting for cache control block
Wait-CXT	Waiting for context block control
Wait-USR	Waiting for directory services control
Wait-IO	Waiting for I/O to complete
Wait-TTO	Waiting to do terminal output
Wait-SHF	Waiting for shared file locked block
Wait-SMB	Waiting for system message buffer
Wait-SPF	Waiting for free blocks in spool file
Wait-TI	Waiting for terminal input
Wait-SDD	Waiting for special device control
Wait-SPC	Waiting for spool file control block
Wait-MSG	Waiting for message from message channel
Suspended	Waiting for .RSUM or completion routine
Wait-TMR	Waiting for .TWAIT expiration
Wait-MEM	Waiting for memory expansion

9 Improved modem control for phone lines

Three new parameters (ONTIM, TIMIN, TIMLOC) have been added to TSGEN to improve the degree of control for modem lines. Previously all modem control was provided by the TIMOUT and OFFTIM parameters. TIMOUT caused a phone job to be killed if the carrier signal was absent for this period. However, it controlled both the time allowed to establish the initial connection and the time an existing connection could be broken. The OFFTIM parameter was used to hang up the phone (by lowering the DTR signal). However, it controlled both how much time was allowed to initiate the job and how long after the job logged out before the phone was hung up.

Modem control is only applied to time-sharing lines defined as phone lines either by the $PHONE flag in the FLAGS macro in the line definition block or by the SET TT n PHONE command. All the following discussion applies *only* to phone lines. Also note that behavior of phone lines and modem control depends on numerous other factors, including the interface card, wiring, and modem. See the *TSX–Plus System Manager's Guide* for more details on TSX-Plus support of modem control.

Modem control consists of: monitoring the ring signal (RI), answering the phone (by raising DTR), monitoring the status of the connection (the carrier signal — DCD), hanging up the phone (by lowering DTR), and killing the job if appropriate. All phone lines are monitored at 0.5 second intervals, and all phone related time intervals should be specified in 0.5 second units. The TIM*xxx* parameters are used to log off an active job if carrier is absent for a specified interval. The *xxx*TIM parameters are used to lower DTR (hanging up the phone) on inactive lines.

The TIMIN parameter is used to allow a normal modem connection to be established after a phone rings and DTR is raised to answer it. If carrier is not established within the TIMIN interval, then DTR is immediately dropped for inactive jobs, and active jobs are automatically logged off (DTR is then dropped after OFFTIM). This parameter should normally allow enough time for the connection to be established and reported at the remote end and for the remote to activate the job — usually slightly longer than the ONTIM interval (see below).

The TIMOUT parameter is used to monitor lost carrier signals during a normal phone session — carrier was established and the job was activated. If carrier is continuously absent for the TIMOUT period, then the job will be killed. If this happens, then DTR will be dropped after the OFFTIM interval. If carrier is re-established within the TIMOUT interval, then the timer is again reset to TIMOUT. If this parameter is too large, then it is possible for the phone connection to be broken without logging off and leave the job active. If another modem dialed in within that time, then it could become connected to the existing job. Except in cases where occasional carrier losses are expected because of noisy line conditions this parameter should be kept small (1-2 seconds).

The TIMLOC parameter is used in conjunction with the PHONE parameter to control local use of phone lines. Phone lines may optionally be used as local lines if carrier is absent when the line is initiated, depending on the PHONE parameter. If PHONE is set to 0, then phone lines may be logged on as local jobs and will not be killed if carrier is not established. If PHONE is set to 1, then the TIMLOC parameter is used to determine how long the job may remain active without establishing carrier. (If carrier is already present when a line is activated, then it is always

monitored and controlled by the TIMOUT parameter.) This parameter should normally specify a short time.

The ONTIM parameter is used to control lowering of DTR after a ringing phone has been answered. This determines how long the modems have to make a connection (establish carrier) and activate the line. Once a line is activated this parameter becomes irrelevant. In order to allow for line noise and automatic speed adjustments between modems, this should usually be a fairly long time (15-30 seconds). The ONTIM parameter is also used to determine the amount of time allowed to enter the correct system password on lines which use it.

The OFFTIM parameter is used to control lowering of DTR when a line logs off or is killed. (Note that DTR may also be lowered after TIMIN if the line is never activated.) In the case of a normal modem connection, it is sometimes desireable to log off and then immediately log back on (perhaps as another account) to reset the environment. This parameter should normally allow only enough time to accomplish this. If this parameter is too long and the phone connection is not broken then the line will be unavailable for other dial-in sessions.

Monitor SET commands may be used to alter any of these parameters. However, they will not affect a line for which the particular timer is currently active. A monitor set command has also been added to change the PHONE parameter. If this value is changed, it takes effect immediately.

A monitor SHOW command has been added to display the values of all the modem related system parameters.

```
.SHOW MODEM
PHONE=1     Other modem control parameters (in 0.5 sec):
 ONTIM=120 OFFTIM=4 TIMIN=120 TIMLOC=1 TIMOUT=10
```

10 Miscellaneous changes

10.1 Keyboard monitor

1. The SET TT VT200 command automatically disables 7-bit masking, an implicit SET TT 8BIT. However, returning the terminal to VT100 or VT52 mode did not clear the 8-bit flag. Both SET TT VT100 and SET TT VT52 now cause an implicit SET TT **NO**8BIT.

2. The SET TT command has been enhanced by the addition of the SET TT VT2207 and SET TT VT2208 commands. These commands allow you to inform the system that the VT220 is set for 7-bit or 8-bit controls. This does not affect either the SET TT [NO]8BIT or the SET TT bits={7 | 8} setting.

3. The SET TT command has been enhanced by the addition of the SET TT STANDARD (or STD) and the SET TT ALTERNATE commands. In the standard mode, exclamation marks are not passed through to a .GTLIN request and control–C does not activate a .GTLIN request. In alternate mode, exclamation marks are passed through to a .GTLIN request and control–C does activate a .GTLIN request. Note that exclamation marks are never passed

to a .GTLIN request made by a program which was started from within a command file, in either standard or alternate mode. The alternate mode is intended for use with interactive programs which use the exclamation mark as a comment character and perform their own control–C handling, such as BASIC–Plus[1].

4. The SET CL command has been extended to allow highly privileged users to clear CL units which have been hung by an XOFF from the remote end even if the unit has been allocated or is in use by another job. These extended CL commands are: SET CLn XONBYPASS and SET CLn RESETBYPASS. The additional BYPASS extension is required to distinguish them from the normal SET CLn XON and SET CLn RESET commands which use the LOOKUP/SPFUN/CLOSE method. The extended versions perform the same functions with regard to the CL unit, but differ in their protection mechanisms as they use the EMTs described in section 10.2 item 8.

5. The SET PRINTWINDOW command has been enhanced by the addition of two new options: /[NO]FLAG and /[NO]STAMP, to disable/enable FLAG pages and/or date/time stamps. Enabling printwindow flag pages does not enable flag pages for a spooled device, only allows a flag page to be generated if the spooled device already has flag pages enabled. Flag pages are ignored for non-spooled devices. The date/time stamp may be used either instead of or in addition to flag pages, and may also be used when the selected printwindow output device is not spooled (e.g. disk files).

6. The SHOW MEMORY command now includes the high limit address of the "Size of unmapped TSX and handlers". This is the portion of the system which is constrained to be less than 40Kb. Since the size in Kb is rounded up, it was frequently difficult to know how much room was left for additional lines or other features. Display of the high limit allows much better estimates of what additional features may be included. The high address, displayed in parentheses, is an octal value which must not exceed 117776. For example, in the following situation:

```
Size of unmapped TSX and handlers = 40Kb  (117660)
```

   the high limit is 117660. Subtracting this from 117776 shows that only 116 octal (78 decimal) bytes are available for additional features.

7. The SHOW REGION command now displays the base of named global regions. The value displayed is the octal equivalent of the address of the base of the region divided by 100 (octal). This is the region base's 64-byte (32-word) block number.

8. The SHOW TERMINALS command now always shows phone lines as type "Phone" when the system PHONE parameter is set to 1. Previously, it was possible for phone lines which were temporarily logged on as local lines (carrier was not present when the line was initiated) to be identified as "Local".

[1]BASIC–Plus is a product of Digital Equipment Corporation

9. The SPOOL *ddn*,STATUS command now displays whether or not the spooled device is set to print a flag page for each file.

10. The left and right parenthesis characters ("(" and ")") now behave as word delimiters for the single–line editor. This means that they serve as character deletion stopping points for the line–feed key and in "KED" mode for the word–delete key.

10.2 System service calls (EMTs)

1. The EMT to enable or disable spooler "hold" mode for a file (EMT 375, function 151) now returns the previous state in R0. (R0 equal 0 indicates previous "nohold" mode; R0 not equal to 0 indicates previous "hold" mode.) This allows the program to reset the spooled device to its initial state before exiting. If this EMT is erroneously issued for a non–spooled device, then on return from the EMT, R0 will contain a negative value (the high bit will be set).

2. An EMT is now available to turn on or off spooler flag pages. The form of the EMT is:

```
EMT     375
```

   with R0 pointing to the following argument block:

```
.BYTE   chan,151
.WORD   1
.WORD   flag
```

   where *chan* is the number of the channel that has been opened to the spooled device, and flag indicates whether or not flag pages are to be enabled. If *flag*=0 flag pages are disabled; if *flag*=1 flag pages are enabled. This EMT must be issued *after* a channel has been opened to the spooled device, but *before* anything is written to it. On return, R0 will contain the previous "flag" setting for the spooled device. (R0 equal 0 indicates previous "noflag" state; R0 not equal 0 indicates previous "flag" state.) If the channel is opened to a non-spooled device, on return R0 will contain a negative value (the high bit will be set). If the specified device is not found, the carry bit will be set on return from the EMT.

3. An EMT is now available to set flag pages to center on 80 or 132 columns. The form of the EMT is:

```
EMT     375
```

   with R0 pointing to the following argument block:

```
.BYTE   chan,151
.WORD   2
.WORD   wide
```

where *chan* is the number of the channel that has been opened to the spooled device, and wide indicates whether flag pages are to be centered on 80 or 132 columns. If *wide*=0 flag pages are set to 80 columns wide; if *wide*=1 flag pages are set to 132 columns wide. This EMT must be issued *after* a channel has been opened to the spooled device, but *before* anything is written to it. On return from this EMT, R0 will indicate the previous state of the "wide" or "narrow" setting. (R0 equal 0 indicates "narrow" state; R0 not equal 0 indicates previous "wide" state.) If the channel is opened to a non-spooled device, on return R0 will contain a negative value (the high bit will be set). If the specified device is not found, the carry bit will be set on return from the EMT.

4. An EMT is now available to determine the number of blocks in the spool file that are currently in use. The form of the EMT is:

```
EMT     375
```

with R0 pointing to the following argument block:

```
.BYTE   1,107
```

The number of spool blocks currently in use is returned in R0. If the spooler was not included at system generation then zero is returned in R0. No error codes are returned.

5. An additional sub-function has been added to the EMT to obtain the TSX–Plus site incremental license number so that the site name may also be obtained. Because of the implementation, this EMT is only available when the spooler is included during system generation. If the spooler has not been included, the EMT will immediately return with the carry flag set.

   The form of the EMT to obtain the site incremental license number (last 5 digits of the full TSX–Plus license number) is:

```
EMT     375
```

   with R0 pointing to an argument block of the form:

```
.BYTE   0,124
```

   On success, the incremental license number is returned in R0.

   In order to obtain the site name string, the argument block should be of the form:

```
.BYTE   1,124
.WORD   buff-ptr
```

   where *buff-ptr* is the address of a **word-aligned** buffer to hold the returned site name. The buffer should be at least 46 bytes long. The actual site name length is returned in R0.

   If the carry bit is set on return from either of these EMTs the system was generated without spooling or the buffer address was odd (site name only).

6. A new option has been added to the job status information EMT to allow the determination of both primary and parent processes for any job. The form of the EMT is:

```
EMT     375
```

with R0 pointing to the following argument block:

```
.BYTE   0,144
.BYTE   line-num,10.
.WORD   buf-address
```

where *line-num* is the number of the job about which information is to be returned. Line numbers are in the range 1 up to the highest valid line number for the system. *Buf-address* is the address of the first word of a 2-word buffer area. The job number of the primary process is returned in the first word of the buffer area, and the job number of the parent process is returned in the second word of the buffer area.

The results of the EMT are explained as follows:

Type of job specified by *line-num*	**Primary** (word 1)	**Parent** (word 2)
Primary	self	0
Virtual	primary	primary
Detached (command)	self	command issuer
Detached (start-up)	self	0

If an error occurs during the execution of the EMT, the carry-flag is set on return and the following error codes indicate the type of error:

Error Code	*Meaning*
0	Specified line number is not currently logged on.
1	Invalid sub-function code.
2	Invalid line number (0, or higher than largest valid number).

7. The EMT to convert a virtual job address to a 22–bit physical address has been split to handle separate I– and D–space addresses. The original form (0,140) checks for special I–space mapping. The new form (22,140) checks for special D–space mapping. If the job has not enabled separate I– and D–space mapping, then both forms are identical. If there is no special mapping for the specified address, then the virtual address is treated as a simple offset from the physical base of the job. This means that, for either form of the EMT, when the specified address is above the valid top of the job or is a PAR7 address mapped either to simulated RMON or to the I/O page, then the EMT will return without error but the value will probably be erroneous.

The form of the EMT to convert a virtual D–space address to a physical address is:

```
EMT     375
```

with R0 pointing to an argument block of the form:

```
.WORD   22,140
.WORD   virtual-address
.WORD   result-buffer
```

8. Two new system services have been defined to perform CL device control. They are equivalent to the special functions to clear and reset a CL unit (.SPFUNs 201 and 265 respectively). The advantage of performing these functions with EMTs is that, unlike .SPFUNs, no channel has to be opened to the specified CL unit.

   Using the EMT forms rather than the SPFUN forms can be particularly useful when dealing with busy spooled CL units connected to a printer or other device which needs to be cleared. If the device is busy enough that all system spool file control blocks are in use, then the SET CLn XON or SET CLn RESET commands will hang, waiting for a free spool file control block, because they have to open a channel to the spooled device to issue the clear or reset special functions. This becomes a deadlock condition when the device cannot proceed because of an XOFF condition, so files are never completed and the spool file control blocks are never released, so the command cannot proceed to clear the device, so the device cannot proceed and ... .

   Another application of these EMTs is allow another job to clear a terminal/CL cross connection (SET HOST) which has received an XOFF and the terminal output buffer is full. When this happens, no more characters can be accepted from the keyboard, including the keystrokes necessary to either break the cross connection (^\) or to clear the XOFF condition (^AR). As with jobs using VTCOM, cross connected lines should allocate the CL unit to prevent communication conflicts. This allocation ordinarily prevents any other jobs from resetting the CL unit, although a subprocess of the cross connected job would not suffer from allocation conflict. However, because of the very low processing level at which the cross connection is serviced (for speed), the job cannot switch to a sub-process. Thus, the job cannot proceed or clear itself and formerly neither could anyone else, although as a last resort the job could be killed. Since these EMTs do not open a channel to the device, normal device allocation checking is bypassed. However, as a measure of protection to prevent hostile use of these functions, they have been implemented to require both TERMINAL and BYPASS privileges.

   The form of both EMTs is:

```
EMT     375
```

   with R0 pointing to an argument block of the form:

```
.BYTE   n,155
.WORD   cl-unit-number
```

where *n* is 1 to clear the CL unit (clear XOFF received flag and transmit an XON) or 2 to reset the CL unit (empty the input silo and output ring buffer, stop sending any break, clear XOFF received flag, send an XON, clear end-of-file status, and reset line and column numbers); ***cl-unit-number*** specifies the CL or C1 unit number – CL0 is 0, CL3 is 3, C10 is 8, C17 is 15, etc.

These EMTs return the following error codes:

Error Code	*Meaning*
0	Invalid EMT subfunction code
1	Job issuing request does not have TERMINAL privilege
2	Invalid CL unit number specified
7	Specified CL unit is not assigned to a line
8	Job issuing request does not have BYPASS privilege

9. The .SERR/.HERR requests now return in R0 the previous state of this request. If user error handling was in effect (.SERR), then R0 will contain a one. If user error handling was not in effect (.HERR), then R0 will contain a zero.

10. Code was added to the execution of device handler time-out routines to clear cells (the link word and the completion address) in the time-out block.

11. A minor change has been made to the .LOOKUP request for non-RT–11 special directory structured devices (notably magtape). Formerly, when the device was off–line a .LOOKUP operation returned an error code of 375 (.SERR code -3; directory I/O error) and the channel status word was cleared. Now, for special directory structured devices, only the ACTIV$ bit is cleared in the CSW for the channel. The .LOOKUP will still fail with error code 375; however, rather than clearing the entire CSW, only the "channel in use" bit will be cleared. The purpose of this modification is to allow the .CSTAT request to identify the device and return device name and unit information. This is required in order to support the IND .TESTDEVICE command.

10.3 System internals

1. The table of device handler block one locations is now maintained in simulated RMON. This is the location on the boot device of the handler file (SY:dd.TSX) plus one. Pseudo-devices which do not actually have handler files (like TT and CL) will have the value zero in this table. Prior to this version, these values were either zero for pseudo-handlers or one to indicate that the handler was loaded.

2. Alterations were made to the UNIBUS system code which allocates and initializes UNIBUS Mapping Registers (UMRs) for extended UNIBUS DMA requests. This alteration provides 18-bit UNIBUS address from the $MPPTR system interface for addresses which have been

physically mapped by the user into the UNIBUS address space (from 3840 to 4096 Kb). This does not affect normal existing code, but does provide additional functionality for interfacing real-time programs with UNIBUS DMA requests.

3. Two new system start-up error messages have been added. The first, "TSX-F-Invalid CSR for device: *ddn*", is reported when there is no response from the CSR for device *ddn*. The second, issued when the handler installation code for device *ddn* fails, is "TSX-F-Error executing installation code for device: *ddn*".

4. A new macro has been added to TSGEN to control the "hold" setting for individual spooled devices. In the previous implementation, the sixth parameter to the SPOOL macro was used to enable or disable the hold setting for all spooled devices. If it is present, the sixth parameter to the SPOOL macro behaves as it has in the past. However, if it is left blank, then the new SPHOLD macro may be used to control the initial hold/nohold setting for each spooled device individually. The SPHOLD macro accepts as many parameters as there are spooled devices. Each parameter may be either "H" or "N" to specify hold or nohold for the respective device. For example, the following macros:

```
SPHOLD  H,N

SPOOL   2,20.,3,2000.,<LP CL2>,,5.
```

   specify that spooled device LP will hold files from being printed until the channel is closed, and spooled device CL2 will allow files to start printing as soon as they are created.

   The keyboard command:

   SPOOL *ddn*,[NO]HOLD

   may be used to dynamically change the hold/nohold status for each spooled device.

5. The algorithm by which the system version and update values in simulated RMON are determined has been changed. Previously, the version and update were always obtained from the running version of RT–11 from which TSX–Plus was started. Now, the version is ordinarily obtained from RT–11, but is limited to the most recent version and update of RT–11 with which the TSX–Plus version is compatible. Selection of the correct version number is important for determining if the RT–11 version will support certain handlers (DU, XL, and MU), and to insure the proper behavior of certain utilities — mostly VTCOM and IND. In particular, the format of the LD translation tables changed at RT–11 version 5.4 and the version of the XL handler which is emulated by CL must match the correct version of VTCOM. The LD and CL versions are determined according to a table consulted during system initialization based on the version of RT–11.

   If the version obtained from RT–11 is recognized and not greater than the last compatible RT–11 version, then that value is copied to offsets 276 (SYSVER) and 277 (SYSUPD). If the version obtained from RT–11 is recognized, but newer than the latest compatible RT–11 version or if the version is unrecognized, then the default version is used.

10.4 TSXMOD

1. The parameters NCSILO, NCXON, and NCXOFF are now modifiable by TSXMOD. Although they were documented as being changeable in version 6.3, the change was omitted from the final release.
2. The parameter PMSIZE may now be adjusted with TSXMOD.
3. The SHOW command has been improved to allow minimum abbreviations for all parameters.
4. When TSXMOD is invoked from a command file, the revision date is only shown once instead of after every blank line.

10.5 PRO/TSX–Plus

1. Changes were made to reduce the program size of TSX.SAV to make it easier to start under the distributed RT–11 FB monitor. It is still necessary to unload device handlers (usually DZ) not required to start TSX.
2. The number of extra CL lines included in PRO/TSX–Plus has been increased from 4 to 6. This allows all non–console ports to be used simultaneously for printers, plotters, modems, etc.

11 Corrected problems

11.1 Keyboard monitor

1. When the system was booted and run from a device unit other than zero (e.g. DU1 or FW2), then the $STOP, $SHUTDOWN, and BOOT commands might not cause the system to reboot to RT-11.
2. $STOP from a primary line which has active subprocess(es) no longer issues the message, "You have 1 active subprocess. Are you sure you want to log off (Y/N):".
3. BOOT with a logical device name will now boot to the assigned physical device.
4. The R[UN] command did not apply installed program attributes and privileges when a device was specified without a unit number. Note that the INSTALL command equates no unit number to unit 0. Now, for example, RUN DM:TEST is equivalent to RUN DM0:TEST and will acquire the correct installed attributes and privileges.
5. When a program was run using the /SCCA switch or installed with the /SCCA option to inhibit control–C aborts, the program could be aborted by typing a control–C when the program requested input.
6. The SET LOG FILE=*dev:file.ext* command now requires a file name if the specified device is directory structured. This prevents creation of .LOG files with no file name.

11.2 System service calls (EMTs)

1. A problem in the .CTIMIO code used by device drivers to cancel pending I/O requests could have caused time-out requests which were not really active to incorrectly cancel a pending time-out request.

2. When a region with active PLAS windows was eliminated, the windows were correctly unmapped, but they were not returned to the available pool of windows. Now, any active windows associated with a region are automatically eliminated when the region is eliminated.

   When active windows are automatically unmapped as a consequence of a .ELRG request, the RS.UNM bit is now returned in the region status word (R.GSTS). Also see the documentation note in Section 12 at item 1.

3. Programs could abort with the error "Invalid EMT argument" when closing a channel to a special directory structured device (e.g. magtape) when the immediately preceding EMT had certain values in the second word of its argument block.

4. If a .CLOSE request was issued to a non-RT–11 file structured device (like magtape) and the previous I/O request to the device had generated an unreported error, then the .CLOSE would fail with "?MON-F-Directory I/O error" (.SERR –3). This could happen when doing asynchronous (non-wait mode) I/O to the device; the error would then be reported by the next I/O operation. Since directory operations (like .LOOKUP and .CLOSE) on non-RT–11 file structured devices are queued to the handler, they would report the previous error as a directory I/O error, which was somewhat confusing. This could have been prevented by issuing a .WAIT prior to the close — this is still recommended practice. Waiting for previous I/O to complete and accepting any consequential error report is appropriate prior to closing the channel.

11.3 System internals

1. When a large number of terminals were generated, the system could have exhibited random failures during start-up, including kernel mode traps, traps to 4, and failure to initiate time-sharing lines. This was due to values residing above the 40000 address (in TSEXEC) which were referenced by KMON.

2. Subprocesses were incompletely initialized and exhibited bizarre behavior if the system was unable to copy the primary line key definitions. This could happen when the TSGEN parameter SEGBLK was set to a small number.

3. When windows were enabled for a terminal in VT200 mode and VT52 emulation was enabled on a subprocess line, on return to the primary line the terminal would be set in VT100 mode rather VT200 mode.

4. When windows were enabled for a terminal and the VT100|200 escape sequence to unlock a terminal's keyboard (ESC[2l) was written to the terminal, TSX–Plus would behave as if the

escape sequence to force VT52 emulation (ESC[?2l) had been the sequence written to the terminal. This caused the system to then incorrectly identify the terminal as a VT52.

5. The system dump facility might not have executed when required if the value specified for the TSGEN parameter DMPTCR was not exactly the address of the receiver or transmitter status register for the dump device. As before, either the receiver or transmitter status register may still be specified for DMPTCR. The automatic compensation to use the dump device transmit CSR and buffer is now more robust.

6. The .GVAL to get the command file status word in simulated RMON (366) did not always return the correct status. A direct reference to offset 366 of simulated RMON (not using a .GVAL) still may not return the correct value.

7. Users could gain unauthorized access to logical disks on the system device.

11.4 SYSMON

1. SYSMON now uses the lead–in character defined in TSGEN (TSLICH) rather than always using the default value of 35 octal.

2. The SYSMON user times and CPU modes displays would not always display a percentage line when the percentage for that particular item was 100%.

11.5 CCL

1. The BACKUP command did not allow two input file specifications, which prevented use of tapes with more than one file or volume (multiple savesets). Note that multiple savesets were introduced in RT-11 V05.04B and correction of the BACKUP command will still not enable you to use BUP savesets unless you have that version or later.

2. The BACKUP command did not correctly handle query for both input and output specifications.

12 Documentation additions and corrections

1. On return from the .ELRG routine, the first and third words of the user's region definition block (R.GID and R.GSTS) are cleared. This change was made in version 6.30, but was not documented.

13 Known problems and restrictions

1. There is a discrepancy between RT–11 and TSX–Plus activation on CTRL–Z and CTRL–C typed in response to the command string interpreter (.CSIGEN, .CSISPC, .GTLIN). However, see the description of the SET TT ALTERNATE command.

2. RT–11 allows a .MAP EMT with a region id equal to zero; TSX–Plus requires that a valid region id (returned from a .CRRG) must be specified for the .MAP EMT to function.

3. RT–11 allows the .CRRG macro to specify a region base address (RS.BAS). TSX–Plus does not allow this.

4. If an IND control file is installed with special privileges, the privileges are lost after execution of a program or keyboard command. This can be circumvented by installing an ordinary command file with the desired privileges which then invokes the IND program.

5. If the program controlled terminal option N is used to suspend command file input and a .GTLIN is done, then command file input cannot be restored. This sequence is usually used to require operator response — in this case, use the optional *type* argument to the .GTLIN request to force terminal input rather than disabling command file input.

6. If the device from which a command file is being read is squeezed during execution of the command file, then the command file pointer will be invalid and unpredictable behavior will result. The behavior depends on the contents of the disk after the squeeze at the location of the command file prior to the squeeze. This is yet another good reason for mounting devices prior to using them. The SQUEEZE command checks that no other users have mounted or allocated the device and that no other jobs have I/O channels open to the device.

7. Programs which corrupt data in the simulated RMON area can receive erroneous error messages and may hang until the system is rebooted.

8. If so many jobs are locked in memory that not enough user memory is available to start another job, then attempting to switch to a subprocess can permanently "freeze" the line.

9. The break sentinel character does not activate entry to the completion routine if the job is already in terminal input state (TI).

10. If a log file is opened and then many characters are displayed via .TTYOUT requests from within a completion routine, the job may hang.

11. Expiration of repeated .MRKT requests delays expiration of an outstanding .TWAIT request.

12. Use of a function key past the end of a field width limit can leave an escape character in the input buffer.

13. Field width limit is ignored for command file input.

14. If two messages are sent at the same time to one job waiting for message completion, the message completion routine is entered twice with the last message which was queued.

15. If a UCL command is defined to be a list of commands and one of those commands is itself a UCL command, none of the commands listed after the inner UCL command will be executed when SET UCL LAST is in effect.

16. The maximum number of Process Windows is limited by the space left for window control blocks in the system window manager overlay. Currently the limit is 24 total windows.

17. Process Windowing does not currently support fields protected against selective erasure. (Control sequence: CSI Ps " q).

18. Process Windowing does not support control sequences unique to the VT3xx series of terminals.

19. When .ABTIO is issued under RT–11, R5 contains the address of the channel status word (CSW) on entry to the device handler abort code. Under TSX–Plus, the CSW resides in user job space and is not directly accessible by device handlers. For this reason, device handler abort entry code should compare with Q.UCSW rather than Q.CSW in order to implement abort requests on specific channels.

20. When .ABTIO is issued under TSX–Plus, pending I/O operations which contain completion requests will be queued as normal I/O completions with the hard error bit set in the CSW. Note that the error will be reported on the first operation to the channel following an .ABTIO.

21. The TSGEN parameter SEGBLK is restricted to be less than 4096 (less than 2 megabytes) due to the implementation of the cache control tables and the PDP–11 architectural design of 8 Kb per mapping register.

22. Programs which use .SPFUN requests to do absolute track and sector reads or writes on single density floppies via the DY handler should not use I/O mapping. In particular, when using single density interchange diskettes with FILEX in an RX02 drive, the FILEX program must be locked in low memory to bypass the need to map the transfer through the system I/O buffers.

23. Professional Series Computers:

    (a) Using defined keys after setting the Pro console (PI handler) into VT220 mode may cause Pro/TSX–Plus to hang or crash. Do not send the sequence to enable VT220 mode (<ESC>[?39h) to the Pro console.

    (b) Sending the terminal reset sequence "<ESC>c" to the Pro console may cause Pro/TSX–Plus to hang or crash. Do not send this sequence or use the SETUP RESET command on the Pro console. SETUP options which are not supported and should not be used on the Pro console are listed in the Pro/TSX–Plus Installation Guide.

    (c) The SETUP utility is not supported on the Professional. See the PRO/TSX-Plus SETUP supplement for more details.

Index