RSX

MULTI-TASKER

January 1985 Issue

RSX Multitasker

Table of Contents

From the Editor

This is my first issue as sole editor of the Multitasker. I would like to thank Allen Watson for a year of hard work as Multitasker co-editor. We all send AL our best wishes in his future endeavors.

In this month's issue there is an excellent article from the RSX development team on TKB internals. For those using FORTRAN-77, Ask a Question may be of interest.

As always, the Multitasker is looking for articles, columns, suggestions, etc. from our readers. Please send submissions directly to me at the address below.

Happy New Year!

Dominic DiNollo
Loral Electronic Systems
Engineering Computer Center
Ridge Hill
Yonkers, New York 10710

Hints and Kinks

Does anyone out there know how to get around the problem of having too many parameters on a FORTRAN "OPEN" statement ? I am trying to open an indexed file with 7 keys, but when I FORTRAN-77 compile, I get error 11-F.

Thanks for your time and help.

Response from Ed Cetron

The solution is to increase the expression frame analyzer stack within the compiler. Read the .cmd file for f77 build on B-2 in the Installation Guide/Release Notes. If you make the line:

```
extsct=stack1:1160
```

something like extsct=stack1:3000
the problem should go away after you re-link the compiler.

Introduction to the Task Builder Internals

Digital Equipment Corp.

1.0 TASK BUILDER INTERNALS

This appendix is a brief overview of the RSX-11M/M-PLUS Task Builder internal operation. It assists in the maintenance of the Task Builder as well as provides the necessary aid to the development process. This appendix gives you a detailed description of the basic flow of processing that the Task Builder (TKB) performs while generating task images, resident libraries and shared common areas.

2.0 BASIC PHASES OF OPERATION

The operation of the Task Builder (TKB) is best described as separate phases. These phases are:

- o Phase 1 - Initialization
- o Phase 2 - Option Processing
- o Phase 3 - First Module Scan
- o Phase 4 - Memory Allocation
- o Phase 5 - Task Image and Map File Generation
- o Phase 6 - Symbol Table File Generation

The phases are described in detail in the following sections.

3.0 PHASE 1 - INITIALIZATION

TKB performs initialization first, which the next sections describe.

3.1 Data Structures

TKB allocates data structures out of the dynamic memory area at addresses higher than and just above the task image and initializes the virtual memory system. The virtual memory system consists of the dynamic memory management routines, which control the allocation and deallocation of blocks of real memory, and the virtual memory management routines, which manages the storage and retrieval of dynamic memory pages on disk. Hereinafter the virtual memory system is referred to as VMS.

The system library routines listed next compose the dynamic memory management portion of VMS. These routines are:

- o $INIDM - Initialize dynamic memory
- o $RQCB - Request core block
- o $RLCB - Release core block

The virtual memory management routines make up the remaining portion of the VMS and consist of four general classifications. They are listed below with the modules that comprise each of the four categories:

  - $INIVM-Virtual memory initialization routine

- o Core allocation routines
  - $ALBLK - Allocate core block
  - $GTCOR - Get core block
  - $EXTSK - Extend task
  - $WRPAG - Write page
- o Virtual memory allocation routines
  - $ALVRT - Allocate virtual memory block
  - $ALSVB - Allocate small virtual memory block
  - $RQVCB - Request virtual core block
- o Page management routines
  - $CVLOK - Convert to real address and lock in dynamic memory

- $CVRL - Convert virtual address to real address in dynamic memory
- $RDPAG - Read virtual page into dynamic memory
- $FNDPG - Find selected page in virtual memory
- $WRMPG - Write-mark selected page in dynamic memory
- $LCKPG - Lock page in dynamic memory
- $UNLPG - Unlock page in dynamic memory

When the VMS is initialized, the virtual memory management routine $INIVM opens an unnamed work file such that when it is closed, it will be deleted. Each time the VMS needs more space, the file is extended by 31 blocks until it reaches the theoretical maximum size of 256 blocks.

3.2 Command Line Processing

TKB obtains your input command line by means of the GCML$ directive and parses the line. TKB allocates an element descriptor from dynamic memory for each output file and stores the switches associated with that file in an output record block associated with each of the output files. The output record block is a unique descriptor containing FCS related information necessary for I/O operations. ;.INDEX R$SWTH The element descriptor is 40 (decimal) bytes long and describes each output file. In the element descriptor at offset E$LMND, the specified file is described in terms that are file system dependent. This information is used when TKB opens and closes the file. At offset E$LNUM, TKB inserts a -2 value if the element descriptor has been allocated for an output file. Otherwise, its value is a -1 if it describes a library file (.OLB) or it contains a value that represents the maximum number of program sections (.PSECTs) that the input file (.OBJ) defines. The offset E$LSWT contains the switch bits that are associated with input or output files. TKB sets up and records default switch values.

3.3 Overlay Descriptor Language Processing

During the scan of the input file, the presence of the /MP switch governs whether or not TKB must process an overlay structure. Non-overlaid (single-segment) and overlaid (multi-segment) tasks are discussed n ext.

3.3.1 Single Segment Task - If TKB determines that a single-segment task is to be built, processing continues at entry-point $BLDSF where TKB builds the filename blocks for the system library file and the debugging aid, if requested, and allocates an element descriptor to each file. If the task to be built is to use D-space support and a debugging aid, TKB changes the filename to reflect the correct debugger, namely ODTID.OBJ instead of ODT.OBJ.

3.3.2 Multi-Segment (Overlaid) Tasks - If TKB is to build multi-segment tasks, TKB enters a sub-phase that processes the Overlay Descriptor Language (ODL) in three passes.

3.3.2.1 Overlaid Tasks, Sub-phase 1, Pass 1 - $MLSG0 opens and reads the .ODL file a single line at a time. $MLSG0 checks the syntax of the line for validity, decodes each directive and stores it in dynamic memory for use in later processing.

3.3.2.2 Overlaid Tasks, Sub-Phase 1, Pass 2 - $MLSG1 scans pre-processed and syntactically correct overlay description lines in dynamic memory. It parses the ODL lines and generates segment descriptors from them. $MLSG2 creates a linked input file list for each line, and for each file specification it allocates an element descriptor and builds a module name table. Each module name table contains up to eight modules.

3.3.2.3 Sub-Phase 1, Pass 3 - $MLSG2 produces task segment tables from the parsed segment descriptor previously prepared by $MLSG1 and sets the overlay structure linkages. $MLSG2 releases its working storage to dynamic memory because the overlay structure is no longer needed. Processing continues at entry point $BLDSF as in single-segment tasks.

4.0 PHASE 2 - OPTION PROCESSING

Option keywords drive the option processor, which consists of two sub-phases. These are described next.

4.1 Option Processing, Phase 2, Sub-Phase 1

P2OPT performs the preliminary processing of each option keyword. As P2OPT decodes the option, it calls the ancillary routines to perform octal-to-binary and decimal-to-binary conversions and stores the converted values in the common input parameter block $PARM. P2OPT then continues further processing. If the decoded option keyword makes reference to a shared region of any kind, the module P2LBR processes the filespec and allocates an element descriptor to the file. P2LBR links an element descriptor and all shared region references into the element list at the head of the list. Module P2LBR checks the specified file for validity as a shared region reference.

If the file passes the test for validity and was built position independent and was requested with an APR, P2LBR reads the file label block, calculates the APR bias, and updates the addresses. P2LBR then links the library reference into the library reference list pointed to by $LBRHD. If the APR has not been specified, P2LBR links the library into the library reference list. If a non-PIC library has been requested, P2LBR checks the base address of the library for possible address conflicts with other libraries or the task itself.

4.2 Option Processing, Phase 2, Sub-Phase 2

The final pass for option processing occurs after the options have been terminated by either a single slash (/) or a double slash, (//). Module P2POP is called and its primary function is to map all referenced libraries into the task's addressing space. Because the libraries are somewhat fixed in their address windows, the APR bitmaps must be adjusted to reflect the existence of the library or libraries. The first pass at allocating the APRs for the libraries is done for those libraries that are either built non-PIC or have been built PIC but specified in the option line with an APR. The latter is a reserved APR assignment. The entire library list is scanned for all non-PIC or APR-reserved libraries before APRs are assigned to any PIC libraries. It is during the post-option processing that cluster libraries are marked as such. The significant difference between cluster libraries and standard libraries is that all libraries in a cluster can share the same address window and, therefore, the same base starting address.

5.0 PHASE 3 - FIRST MODULE SCAN

During this major phase, TKB reads all the input modules and generates internal symbol tables for each task segment in the allocation. As TKB considers each segment, TKB scans the element list generating section and symbol tables for that segment. The scan continues for all modules contained in that segment. TKB resolves the element module name and constructs the element section mapping table. The element section mapping table contains information that links each program section seen by TKB to a file defined by an element descriptor. This is done to provide a more efficient path through the input element list during code output in a later phase.

5.1 Pre-Processing, Phase 3, Sub-phase 1

During this sub-phase, the subroutine $P3PRE is called from routine $TASKB. $P3PRE initializes the internal symbol tables and program section tables with the following entries:

- o Global segment names
- o Global references
- o Program section entries for:
  - Autoload vectors - ($$ALVC, RO, I, GBL, REL, CON)
  - Supervisor-mode vectors (if requested) - ($$SLVC, RO, I, GBL, REL, CON)
  - Segment tables -

    ($$SGD0, R/W, D, GBL, REL, CON)
    ($$SGD1, R/W, D, GBL, REL, CON)
    ($$SGD2, R/W, D, GBL, REL, CON)
  - Segment return point -

    ($$RTR, RO, I, GBL, REL, CON)
    ($$RTQ, RO, I, GBL, REL, CON)
    ($$RTS, RO, I, GBL, REL, CON)
  - Window descriptors - ($$WNDS, R/W, D, GBL, REL, CON)
  - Region descriptors - ($$RGDS, R/W, D, GBL, REL, CON)

5.2 Module Scan, Phase 3, Sub-Phase 2

During this sub-phase, TKB initializes the overlay structure path list by calling $WSINI. This routine scans the overlay structure and builds the path list that will be used by the virtual memory symbol table search and insert routines. On output, this module sets the following global variables:

- $CRSEG ::- Real (memory) address of current segment being processe d
- $CRVSG ::- Virtual (disk) address of current segment being process ed
- $PATH ::- Address of current segment path list

On output, $WSINI has constructed the segment path list, which contains the following information:

- SGADR-Virtual address (disk) of the segment descriptor
- PATHF-path flag which has a value relative to the current segment being processed. The possible values are:
  - PATHF = 0, current segment
  - PATHF = 1, segment is up-tree
  - PATHF = 2, segment is in a cotree
  - PATHF < 0, segment is down-tree

Entries are made in the following format:

- The current segment
- All entries down-tree
- All entries up-tree
- All cotree entries

Once the search path has been established for the current segment, processing continues after the coroutine $STINP has set up and opened the first module in the segment element list. TKB allocates and links a concatenate d module descriptor into the element list at the first module. The reason for this apparent duplication of data structures is for dynamic memory conservation. After this first module scan, it is no longer necessary to keep the full file specification because the file IDs are sufficient for later processing. For each entity in the element list, the called subroutine $PRCLM reads each object module, but performs operations only on the global symbol directory (GSD) records. With the exception of block types 0 and 6, all other block types are ignored. Of the block type 1 at the start of the GSD, there are nine sub-classifications. They are:

1. Module name
2. Section name
3. Internal symbol
4. Transfer address
5. Symbol declaration
6. Program section name (.PSECT)
7. Version identification
8. Virtual array storage section name (FORTRAN support)
9. Completion routine name for supervisor-mode libraries.

Upon completion of the module scan for the entire overlay structure driven by the input element list, the final scan on the system object library is initiated. The system object library is treated much like an input module, except that now the concatenated module descriptor is a linked list, all elements pointing to the system object library file specifier. The module $P3LBS performs the scan of this library reading the entry point table, searching for the desired entry point name, and then calling on $PRCLM to process the element.

6.0 PHASE 4 - MEMORY ALLOCATION

Phase 4 of the task build performs the memory and disk space allocation for the task. This phase takes as input all the tables constructed during Phase 3 processing. P4MAL processes resident library and shared common regions first followed by each segment in the allocation. P4MAL assigns to each control section and to all relocatable symbols in the segment absolute addresses relative to zero in that segment. If the current task utilizes a supervisor-mode library, P4MAL uses the supervisor-mode listhead to generate the supervisor-mode vectors to which P4MAL assigns absolute addresses relative to zero in the segment in the same manner as for the autoload vectors. Finally, P4MAL allocates the disk space for the task image. In addition, P4MAL also processes the unit modification table that has been set up during Phase 2 option processing by using the ASG keyword and the patch list that set up by the GBLPAT and GBLDEF option keywords. From this information, TKB calculates the task image header size and determines its position on disk relative to the label block.

6.1 Task Header Generation, Phase 4

Once P4MAL completes the memory and disk allocation, patch list processing, autoload vector and supervisor-mode vector generation, P4MAL calls module HEADR to begin writing the task image out to the disk file. It is during this processing by HEADR that sets the various bits in both the label and header blocks that reflect the task type. From the tables built up during previous phases, HEADR checks the address limits and writes them out to disk in the label block. The label block is used by the INSTALL processing program to determine the run-time requirements of the task, such as library requests, initial memory size requirements, and partition assignment. Once installed, the label block is no longer needed. After the label block has been written to disk, HEADR writes out the logical unit block or blocks. There will be one LUN block if the number of units is less than or equal to 128, or two LUN blocks for anything over 129 up to 255. The label block layout is as follows:

- L$BTSK - Radix 50 task name
- L$BPAR - Radix 50 task partition
- L$BSA - Task starting virtual address
- L$BHGV - Highest virtual address in window zero
- L$BMXV - Highest task virtual address
- L$BLDZ - Task load size (32-word blocks)
- L$BMXZ - Task maximum size (32-word blocks)
- L$BOFF - Task offset into partition (32-word blocks)
- L$BWND - Number of windows required (less libraries)
- L$BSEG - Size of resident segment descriptors
- L$BFLG - Task flags word
- L$BDAT - Creation date (year, month, day)
- L$BLIB - Task resident library requests
- L$BPRI - Task priority
- L$BXFR - Task transfer address
- L$BEXT - Task extend size (32-word blocks)
- L$BSGL - Block number of segment length list

- o L$BHRB - Relative block number of header
- o L$BBLK - Number of blocks in label
- o L$BLUN - Number of logical units
- o L$BROB - Relative block number of read-only image
- o L$BROL - Read-only load size (32-word blocks)
- o L$BRDL - Read-only data load size (32-word blocks)
- o L$BHDB - Relative block number of data
- o L$BDHV - Data window 1 high virtual address
- o L$BDMV - Data high virtual address
- o L$BDLZ - Data load size
- o L$BDMZ - Data maximum size
- o L$BASG - Symbolic device assignments

At offset L$BLIB, HEADR places the resident library and common region requests according to the order of specification. For RSX-11M, the maximum number of library requests is 7 and for RSX-11M-PLUS, this number is extended to 15. After HEADR writes the label block and LUN block(s), it writes the task header block. The Executive uses the header block during the task's run-time, as a dynamic storage region unique to that task. The header block contains the initial values for the Processor Status Word, program counter and stack pointer, as well as the addresses in the task's addressing space of any synchronous system traps that may be used by the task. The header block layout is as follows:

- o H$CSP - Current SP word
- o H$DSIZ - Length of header in bytes
- o H$EFLM - Address of event flags and mask word
- o H$CUIC - Current UIC
- o H$DUIC - Default UIC
- o H$IPS - Initial PS word
- o H$IPC - Initial PC word

- H$ISP - Initial SP word
- H$ODVA - ODT SST vector address
- H$ODVL - ODT SST vector length
- H$TKVA - Task SST vector address
- H$TKVL - Task SST vector length
- H$PFVA - Address of power-fail AST control block
- H$FPVA - Address of floating point exception control block
- H$RCVA - Address of task receive AST control block
- H$EFSV - Event flags save word
- H$FPSA - Address of floating point save area
- H$WND - Address of task window blocks
- H$DSW - Task directive status word
- H$FSR - Address of FCS work area
- H$FOT - Address of FORTRAN OTS work area
- H$OVLY - Address of overlay run-time system work area
- H$VEXT - Work area vector extensions
- H$SPRI - Swapping priority difference
- H$NML - Network mailbox LUN
- H$RRVA - Receive-by-reference AST
- H$GARD - Address of header guard word
- H$NLUN - Number of logical units
- H$LUN - Logical unit table

As a final step, if the task being built has memory resident overlays, HEADR writes the segment load list, which contains the length of each memory resident segment in the task, out to the disk file. Also, if a user-mode I- and D-space task is being built, HEADR writes a second copy of the task header out to disk in the D-space portion of the task image. The support routines needed by the module HEADR are as follows:

- $DSALO - Allocation of task-resident descriptors
- $ALALO - Address assignment for autoload and supervisor-mode vecto rs
- $SGALO - Segment physical and virtual memory allocation
- $SYALO - Absolute address assignment to relocatable symbols
- $DKALO - Disk space allocation for the task image

7.0 PHASE 5 - TASK IMAGE AND MAP FILE GENERATION

This phase of the task build process generates the actual image on disk. P5MAP generates the requested map file during this phase. As P5MAP considers each segment in the allocation, TKB scans the element list for each and the scanning continues to the end of the module list. TKB performs relocation and writes the resultant text to the task image disk file.

7.1 Task Image Generation, Phase 5, Sub-Phase 1

First, TKB searches the entire structure for the symbols that are located in branches of the tree structure being built. As TKB locates the symbols, it now resolves their addresses by the offset into the program section $$ALVC. $$ALVC defines the entry point in terms of an autoload vector. Contained in the vector is a subroutine call to the Overlay Run-time System module AUTO ($AUTO), the address of the segment descriptor containing the target routine, and the address of the target routine itself.

The scan of the segments comprising the allocation begins with the root segment. TKB calls module STINP to set up the input stream. $P5ELM processes each object module, which resembles the function of $PRCLM, except that $P5ELM uses the object record types 3 and 4, the text record and relocation directory records. The text record contains the actual code generated by either MACRO-11 or one of the language processors. The relocation directory records govern the processing of the preceding text record and have sixteen legal commands. They are:

1. Type 1 - Internal relocation
2. Type 2 - Global relocation
3. Type 3 - Internal displaced relocation

4. Type 4 - Global displaced relocation

5. Type 5 - Global additive relocation

6. Type 6 - Global additive displaced relocation

7. Type 7 - Location counter definition

8. Type 8 - Location counter modification

9. Type 9 - .LIMIT directive (MACRO-11)

10. Type 10 - Sector relocation

11. Type 11 - Illegal format (Reserved)

12. Type 12 - Sector displaced relocation

13. Type 13 - Sector additive relocation

14. Type 14 - Sector additive displaced relocation

15. Type 15 - Complex relocation

16. Type 16 - Resident library additive relocation

After $P5ELM scans, relocates and writes out to the disk file, sub-phase 1 processing continues with end-of-segment processing. Module P5EOS writes the following out to the disk file: the autoload vectors, supervisor-mode vectors, and the task-resident segment tables for the current segment being processed.

After TKB has traversed and output the entire overlay structure, Phase 5 continues with cluster library processing, if requested by the CLSTR option during Phase 2 option input. Module P5CLS generates and patches task resident segment descriptors, which describe a TKB-generated overlay structure. From the cluster library listhead, P5CLS creates and writes out a dummy root segment to the disk file of the task image. P5CLS then traverses the overlay structures of the clustered libraries making the necessary changes to the segment descriptors, window blocks, and region descriptors for each cluster library element. P5CLS marks each library as such to prevent the INSTALL processing program from flagging the clustered libraries as mapped upon installation. The only exception is the first library in the cluster group, which is the default library. The request for this library is mapped upon installation, the others checked against the common block directory, if running on an RSX-11M-PLUS system, or checked against the partition list for entries associated with the clustered libraries, if running on an RSX-11M system. If any one of the requested libraries is not installed, the task cannot be installed itself.

With the completion of cluster library processing, TKB either compresses and closes or merely closes the task image file. If TKB built either a resident library or a multiuser task, the read-only portion of the task image is compressed. Compression allows easier loading of the read-only part of the image when it is run.

7.2 Map File Processing, Phase 5, Sub-Phase 2

A third pass over the entire overlay structure is necessary if a standard map file is to be generated. Module P5MAP scans the switch word and determines if a map has been requested, and if so whether it is to be a long or short (default) map. If requested, the map file is opened and the map heading is output. If the task being built has an overlay structure, then the overlay structure is written to the map file by module MPOLD. A scan of the overlay structure is initiated to reset the base addresses of each section within the segment currently being processed. If there are any undefined symbols in a segment the symbols are written to the map file as well. There are no other error messages that are written to the map file. Closing the map file completes Phase 5 processing.

8.0 PHASE 6 - SYMBOL TABLE FILE GENERATION

The final phase of the task build is the generation of the symbol table file (.STB) for the task image. It can have two forms depending on whether the task being built is a shared region or a normal task with or without a debugging aid. The differences between the two are discussed in the following sections.

8.1 Shared Region .STB File Generation

The primary function of the .STB file is to provide a means by which the global entry points of a shared region, either a resident library or resident common, resolve addresses generated by subroutine calls within a user task to the resident library or common. The format of the .STB file is identical with that of an object record produced by either MACRO-11 or a language processor. The file contains the same types of records that define the values (addresses) of the entry points in the library or common and any program sections that may be associated with it. Two switches are associated with .STB file processing. They are the /LI and /CO switches.

The use of the /LI switch suppresses the inclusion of all program section entries in a position-independent resident library. This prevents a resident library and the user task from having two program sections with the same name. If this suppression is not done, TKB tries to assign addresses to the program section that has been defined in the library and is fixed in length. TKB then reports a load address out of range error and cancels the task image file generation. In this case, the name of the first module seen in the input stream is used as the library program section name. This ensures, with some measure of certainty, that there is not a .PSECT conflict.

The use of the /CO switch builds a common, which retains all .PSECT information. The /CO switch is the default if /-HD and /PI are both present.

If the resident library is to have an overlay structure, the only symbols that appear in the .STB file are those that have been defined in the root of the library. If there are segments of the library that contain entry point definitions, the symbol must be forced into the root segment through the GBLREF option. This generates an autoload vector entry for that symbol and it is output to the .STB file in the program section $$ALVC. In addition to these symbols, any that have been specified by the GBLINC option are included in the .STB file.

8.2 Normal Task With Debugger .STB File Generation

If the task being built is to have a debugging aid, the .STB file has a slightly different structure. If a symbolic debugging aid is specified, the kernel of the debugger must know the addresses and names of the symbols in the task being debugged. It must also know the overlay structure of the task. For the debugger to maintain this information, TKB must include some sort of one-to-one correlation between the task image and the .STB file that describes it. The correlation is done by an internal symbol definition of the task image creation date and time. This and the creation of internal symbol directory records on a segment-by-segment basis are the only differences between the two types of .STB files (task and shared region). With the generation of this type of .STB file TKB re-scans the entire input module list for each segment.

APPENDIX A

TASK BUILDER INTERNAL DATA STRUCTURES

```
                      Element Descriptor

        +----------------------------------------------------+
        | Link to next descriptor                            |   0 E$LNXT
        |----------------------------------------------------|
        | Real Control Section Mapping Table                 |   2 E$LCMT
        |----------------------------------------------------|
        | Highest section number                             |   4 E$LNUM
        |----------------------------------------------------|
                                                                        | Elen
        |----------------------------------------------------|
        | Version identification                             |  10 E$LIDT
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |----------------------------------------------------|
        | Module name (title block)                          |  14 E$LMOD
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |----------------------------------------------------|
        | Virtual address of Mapping Table                   |  20 E$LVMT
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |----------------------------------------------------|
        | Monitor dependent information                      |  24 E$LMND
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        |- - - - - - - - - - - - - - - - - - - - - - - - - - |
        |                                                    |
        +----------------------------------------------------+
```

Task and STB File Switches

```
         1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
         5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        | | | | | | | | | | | | | | | | |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     1   | | | | | | | | | | | | | | | +-XH$DR -/XH not allowed. (1=yes)
     2   | | | | | | | | | | | | | | +---SW$ALO-Chkpnt space alloc. (0=yes)
     4   | | | | | | | | | | | | | +-----SW$AC -Task is ACP (1=yes)
    10   | | | | | | | | | | | | +-------SW$MU -Task is multi-user (1=yes)
    20   | | | | | | | | | | | +---------SW$NS -Sends not permitted (1=yes)
    40   | | | | | | | | | | +-----------SW$SL -Task is slaveable (1=yes)
   100   | | | | | | | | | +-------------SW$PM -Post-mortem dump (1=yes)
   200   | | | | | | | | +---------------SW$TR -Set trace bit in PS (1=yes)
   400   | | | | | | | +-----------------SW$PR -Task is privileged (1=yes)
  1000   | | | | | | +-------------------SW$PI -PIC output (1=yes)
  2000   | | | | | +---------------------SW$DA -/DA specified (1=yes)
  4000   | | | | +-----------------------SW$CM -Build task /CM (1=yes)
 10000   | | | +-------------------------SW$NH -Build task /-HD (1=yes)
 20000   | | +---------------------------SW$EA -Task uses EAE (1=yes)
 40000   | +-----------------------------SW$FP -Task uses FPU (1=yes)
100000   +-------------------------------SW$CP -Task checkpointable (0=yes)
```

Input File Switches

```
         1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
         5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        | | | | | | | | | | | | | | | | |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     1   | | | | | | | | | | | | | | | +-SW$CLS-Clstr Lib Elem (1=yes)
     2   | | | | | | | | | | | | | | +---SW$SS -/SS applied? (1=yes)
     4   | | | | | | | | | | | | | +-----SW$RL -Res Lib Elem? (1=yes)
    10   | | | | | | | | | | | | +
    20   | | | | | | | | | | | +---------SW$AL -Autoload Element (1=yes)
    40   | | | | | | | | | | +-----------SW$CC -Concatenated file (1=yes)
   100   | | | | | | | | | +-------------SW$SUP-Supervisor-mode library
   200   | | | | | | | | +---------------SW$SV -Super-mode vectors (1=no)
   400   | | | | | | | +
  1000   | | | | | | +
  2000   | | | | | +
  4000   | | | | +-----------------------SW$DA -Module is debugger (1=yes)
 10000   | | | +-------------------------SW$MP -Overlay map file (1=yes)
 20000   | | +---------------------------SW$LB -Library file (1=yes)
 40000   | +-----------------------------SW$DL -File is dflt syslib (1=yes)
100000   +-------------------------------SW$MA -Include file in map (1=no)
```

Control Section Table

```
+-----------------------------------------------+
| Link to next entry                            |    0 C$SLNK
|-----------------------------------------------|
| Section name, radix 50                        |    2 C$SMNE
|- - - - - - - - - - - - - - - - - - - - - - - -|
|                                               |
|-----------------------------------------------|
| Flags word                                    |    6 C$SFLG
|-----------------------------------------------|
| Base address                                  |   10 C$SBSE
|-----------------------------------------------|
| Length of control section                     |   12 C$SLTH
|-----------------------------------------------|
| Address of defining element                   |   14 C$SELM
|-----------------------------------------------|
| Current base address                          |   16 C$SCUR
|-----------------------------------------------|
| Length in segment                             |   20 C$SLGS
+-----------------------------------------------+
```

Flag Word Bit Definitions

```
         1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
         5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        | | | | | | | | | | | | | | | | |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     1   | | | | | | | | | | | | | | | +-CS$ATL-Autoload flag (1=yes)
     2   | | | | | | | | | | | | | | +---CS$LIB-Library section (1=yes)
     4   | | | | | | | | | | | | | +-----CS$ALO-Allocation (1=ovr, 0=con)
    10   | | | | | | | | | | | | +-------CS$IND-Indirect (1=ind, 0=def)
    20   | | | | | | | | | | | +---------CS$ACC-Access (1=R/O, 0=R/W)
    40   | | | | | | | | | | +-----------CS$REL-Relocatable (1=rel, 0=abs)
   100   | | | | | | | | | +-------------CS$GBL-Scope (1=global, 0=local)
   200   | | | | | | | | +---------------CS$TYP-Type (1=data, 0=ins)
   400   | | | | | | | +
  1000   | | | | | | +
  2000   | | | | | +
  4000   | | | | +---------------------CS$SUP-Super-mode library Section
 10000   | | | +-----------------------CS$ROT-Section forced into root
 20000   | | +-------------------------CS$VAS-Virtual array Sec. (1=yes)
 40000   | +---------------------------CS$RES-Task-resident Sec. (1=yes)
100000   +-----------------------------CS$VSC-Virtual Section (1=yes)
```

Symbol Table Entry

```
+--------------------------------------------------+
| Link to next symbol                              |    0 S$YLNK
|--------------------------------------------------|
| Symbol name, radix-50                            |    2 S$YNME
|- - - - - - - - - - - - - - - - - - - - - - - - - |
|                                                  |
|--------------------------------------------------|
| Symbol flags byte                                |    6 S$YFLG
|--------------------------------------------------|
| Symbol value                                     |   10 S$YVAL
|--------------------------------------------------|
| Control Section Entry address                    |   12 S$YCMT
|--------------------------------------------------|
| Address of defining segment                      |   14 S$YSEG
|--------------------------------------------------|
| Value of completion routine                      |   16 S$YCMP
+--------------------------------------------------+
```

Symbol Table Entry Flag Word Bit Definitions

```
         1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
         5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        | | | | | | | | | | | | | | | | |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     1   | | | | | | | | | | | | | | | +-SY$WK -Weak ref. or def. (1=yes)
     2   | | | | | | | | | | | | | | +---SY$ATR-Autoload Vec ref? (1=yes)
     4   | | | | | | | | | | | | | +-----SY$LIB-Symbol def in lib? (1=yes)
    10   | | | | | | | | | | | | +-------SY$DEF-Definition (1=def, 0=ref)
    20   | | | | | | | | | | | +---------SY$ATL-Autoload flag (1=yes, 0=no)
    40   | | | | | | | | | | +-----------SY$REL-Relocation (1=rel, 0=abs)
   100   | | | | | | | | | +-------------SY$GBL-Global (1=global, 0=lcl)
   200   | | | | | | | | +---------------SY$IND-Indirect (1=ind, 0=def)
   400   | | | | | | | +-----------------SY$SAB-Abs def from super-mode lib
  1000   | | | | | | +-------------------SY$SUP-Super-mode def from root
  2000   | | | | | +---------------------SY$SLB-Super-mode def (1=lib)
  4000   | | | | +-----------------------SY$SDF-Super-mode sym decl (1=def)
 10000   | | | +-------------------------SY$RS0-Super-mode future expansion
 20000   | | +---------------------------SY$SRL-Super-mode rel sym (1=rel)
 40000   | +-----------------------------SY$SGB-Super-mode GBL sym (1=GBL)
100000   +-------------------------------SY$EXC-Exc sym from map? (1=yes)
```

Task Builder Internal Segment Descriptor Offsets

Segment status (low byte). High byte is reserved.	0	S$GSTS
Disk block address of segment	2	S$GBLK
R/O disk block address of segment	4	S$GBRO
R/W data disk block address of segment	6	S$GBWD
R/O data disk block address of segment	10	S$GBOD
Virtual load address of segment	12	S$GLDA
Length of segment in bytes	14	S$GLNG
Link up	16	S$GUP
Link down	20	S$GDWN
Link next (link right)	22	S$GNXT
Link previous (link left)	24	S$GPRV
Segment name RADIX-50	26	S$GNME
Virtual address of Window Block Descriptor.	32	S$GWDP
Control Section Table list.	34	S$GCST
Read/write memory allocation in bytes	40	S$GRW
Read/only memory allocation in bytes	42	S$GRO
Read/write data memory allocation in bytes	44	S$GRWD
Read/only data memory allocation in bytes	46	S$GROD
Number of global symbol entries in segment	50	S$GNTB
Symbol Table list.	52	S$GSTB
Count of undefined symbols within the segment	154	S$GUND
Highest virtual address in the segment	156	S$GVAD
Two-word Element Descriptor listhead	160	S$GELT

```
| Supervisor Load list (2 wds), entry count (1 wd)  | 162 S$GSPL
|- - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                   |
|- - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                   |
|---------------------------------------------------|
| Autoload listhead (2 wds), entry count (1 wd)     | 170 S$GATL
|- - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                   |
|- - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                   |
|---------------------------------------------------|
| Highest physical address in segment               | 176 S$GMEM
|---------------------------------------------------|
| Base virtual address of Read/Only root            | 200 S$GROB
|---------------------------------------------------|
| Base virtual address of R/O data root             | 202 S$GODB
|---------------------------------------------------|
| Start of segment Supervisor Mode Vectors          | 204 S$GSUP
|- - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                   |
|---------------------------------------------------|
| Start of segment Autoload Vectors                 | 210 S$GAUT
|- - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                   |
|---------------------------------------------------|
| Start of Region Descriptors                       | 214 S$GREG
|- - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                   |
|---------------------------------------------------|
| Start of Segment Descriptors                      | 220 S$GSEG
|- - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                   |
|---------------------------------------------------|
| Start of Window Descriptors                       | 224 S$GWND
|- - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                   |
|---------------------------------------------------|
| Sequence allocation listhead                      | 230 S$GSEQ
|- - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                   |
+---------------------------------------------------+
```

Segment Descriptor Flags (in S$GSTS)

```
            1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
            5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
           +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
           | | | | | | | | | | | | | | | | |
           +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         1  | | | | | | | | | | | | | | | +-SG$GBL-Seg name is GBL (1=yes)
         2  | | | | | | | | | | | | | | +---SG$PHY-Seg has mem alloc (1=yes)
         4  | | | | | | | | | | | | | +-----SG$RES-Seg is res o'lay (1=yes)
        10  | | | | | | | | | | | | +-------SG$RO -Seg is read-only (1=yes)
        20  | | | | | | | | | | | +---------SG$MEM-Seg in mem? (0=in memory)
        40  | | | | | | | | | | +-----------SG$LOD-Seg ldld in mem (1=yes)
       100  | | | | | | | | | +-------------SG$DSK-Seg has dsk alloc (0=yes)
       200  | | | | | | | | +---------------SG$DES-Seg desc flg (always set)
       400  | | | | | | | +
      1000  | | | | | | +
      2000  | | | | | +
      4000  | | | | +
     10000  | | | +
     20000  | | +
     40000  | +
    100000  +
```

Virtual Memory Page

```
                +-----------------------------------------------------+
                | Link to next Page                                   |    0 P$GNXT
                |-----------------------------------------------------|
P$GSTS     3    | Page Status               | Page Rel Blk _#         |    2 P$GBLK
                |-----------------------------------------------------|
                | 'Time' of last reference                            |    4 P$GTIM
                |-----------------------------------------------------|
                | Page lock count                                     |    6 P$GLOK
                |-----------------------------------------------------|
                | Page Data, 512. bytes                               |   10 P$GTXT
                +-----------------------------------------------------+
```

Page Status Bits

```
                 0 0 0 0 0 0 0 0
                 7 6 5 4 3 2 1 0
                +-+-+-+-+-+-+-+-+
                | | | | | | | | |
                +-+-+-+-+-+-+-+-+
           1     | | | | | | | +-PG$WRT-Page written into
           2     | | | | | | +
           4     | | | | | +
          10     | | | | +
          20     | | | +
          40     | | +
         100     | +
         200     +
```

APPENDIX B

TASK-RESIDENT DATA STRUCTURES

TASK-RESIDENT DATA STRUCTURES

Task-resident Segment Descriptor Offsets

```
+-----------------------------------------------------+
| Disk block address(bits 11 - 0)                     |   0 T$RBLK
|-----------------------------------------------------|
| Virtual load address of segment                     |   2 T$RLDA
|-----------------------------------------------------|
| Length of segment in bytes                          |   4 T$RLNG
|-----------------------------------------------------|
| Link up                                             |   6 T$RUP
|-----------------------------------------------------|
| Link down                                           |  10 T$RDWN
|-----------------------------------------------------|
| Link next                                           |  12 T$RNXT
|-----------------------------------------------------|
| Segment name (2-word radix 50)                      |  14 T$RNME
|- - - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                     |
|-----------------------------------------------------|
| Task-resident window backpointer                    |  20 T$RWDP
+-----------------------------------------------------+
```

Task-resident Segment Descriptor Flags (in T$RBLK bits 15 - 12)

```
          1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
          5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         | | | | | | | | | | | | | | | | |
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      1   | | | | | | | | | | | | | | | +
      2   | | | | | | | | | | | | | | +
      4   | | | | | | | | | | | | | +
     10   | | | | | | | | | | | | +
     20   | | | | | | | | | | | +
     40   | | | | | | | | | | +
    100   | | | | | | | | | +
    200   | | | | | | | | +
    400   | | | | | | | +
   1000   | | | | | | +
   2000   | | | | | +
   4000   | | | | +
  10000   | | | +-------------------------TR$LOD-Seg is loaded mem? (1=yes)
  20000   | | +---------------------------TR$DSK-Seg has dsk alloc (1=no)
  40000   | +-----------------------------TR$MEM-Seg is in memory (1=no)
 100000   +-------------------------------TR$DES-Task-res flag (always set)
```

Task-resident Window Definition Block (WDB)

```
             +---------------------------------------------------+
W.NAPR    1  | Window base APR         | Window ID               |     0 W.NID
             |---------------------------------------------------|
             | Window size in 64 byte blocks.                    |     2 W.NSIZ
             |---------------------------------------------------|
             | Region ID                                         |     4 W.NRID
             |---------------------------------------------------|
                                                                              | Off
             |---------------------------------------------------|
             | Length to map                                     |    10 W.NLEN
             |---------------------------------------------------|
             | Window status word                                |    12 W.NSTS
             |---------------------------------------------------|
             | Send/Receive buffer address                       |    14 W.NSRB
             |---------------------------------------------------|
             | Flags word                                        |    16 W.NLGH
             |---------------------------------------------------|
             | Address of Region Descriptor                      |    20 W$NREG
             +---------------------------------------------------+
```

Window Definition Block Status Word Bit Definitions

```
              1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
              5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
             +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
             | | | | | | | | | | | | | | | | |
             +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        1     | | | | | | | | | | | | | | | +-WS.RED-Send with Read access
        2     | | | | | | | | | | | | | | +---WS.WRT-Send with Write access
        4     | | | | | | | | | | | | | +-----WS.EXT-Send with Extend access
       10     | | | | | | | | | | | | +-------WS.DEL-Send with Delete access
       20     | | | | | | | | | | | +---------WS.BPS-Always bypass cache
       40     | | | | | | | | | | +-----------WS.SIS-Create Super-I window
      100     | | | | | | | | | +-------------WS.RCX-Exit if no RREF$
      200     | | | | | | | | +---------------WS.MAP-Map by CRAW$ or RREF$
      400     | | | | | | | +-----------------WS.64B-Define allowed alignment
     1000     | | | | | | +-------------------WS.NAT-Create Att. Desc.
     2000     | | | | | +---------------------WS.RES-Map only if resident
     4000     | | | | +-----------------------WS.NBP-Do not bypass cache (M+)
    10000     | | | +-------------------------WS.RRF-Reference received
    20000     | | +---------------------------WS.ELW-Window eliminated
    40000     | +-----------------------------WS.UNM-Window unmapped
   100000     +-------------------------------WS.CRW-Window succfully created
```

Task-resident Region Definition Block (RDB)

```
+-----------------------------------------------------+
| Region Identification                               |     0 R.GID
|-----------------------------------------------------|
| Region size in 32w blocks                           |     2 R.GSIZ
|-----------------------------------------------------|
| Region name, RADIX-50                               |     4 R.GNAM
|- - - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                     |
|-----------------------------------------------------|
| Region main Partition name, RADIX-50                |    10 R.GPAR
|- - - - - - - - - - - - - - - - - - - - - - - - - - -|
|                                                     |
|-----------------------------------------------------|
| Region status word                                  |    14 R.GSTS
|-----------------------------------------------------|
| Region protection word                              |    16 R.GPRO
+-----------------------------------------------------+
```

Region Definition Block Status Word Bit Definitions

```
         1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
         5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        | | | | | | | | | | | | | | | | |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     1   | | | | | | | | | | | | | | | +-RS.RED-Read access required
     2   | | | | | | | | | | | | | | +---RS.WRT-Write access required
     4   | | | | | | | | | | | | | +-----RS.EXT-Extend access required
    10   | | | | | | | | | | | | +-------RS.DEL-Delete access required
    20   | | | | | | | | | | | +---------RS.NEX-Region not extendible
    40   | | | | | | | | | | +-----------RS.ATT-Attach to created Region
   100   | | | | | | | | | +-------------RS.NDL-Do not mark for deletion
   200   | | | | | | | | +---------------RS.MDL-Mark for Deletion on detach
   400   | | | | | | | +
  1000   | | | | | | +
  2000   | | | | | +
  4000   | | | | +
 10000   | | | +
 20000   | | +
 40000   | +-----------------------------RS.UNM-Window unmapped on detach
100000   +-------------------------------RS.CRR-Region succfully created
```

Autoload Vector Format

```
+---------------------------------------------------+
| JSR  PC,@_#0                                      |    0 A$UTO
|---------------------------------------------------|
| Address of Autoload subroutine                    |   2 A$ADR
|---------------------------------------------------|
| Target routine Segment Descriptor address         |   4 A$SGA
|---------------------------------------------------|
| Target routine Entry Point                        |   6 A$EPT
+---------------------------------------------------+
```

Supervisor-mode Vector Format

```
+---------------------------------------------------+
| MOVE  (PC)+,-(SP)                                 |   0 S$LVC
|---------------------------------------------------|
| Address of Super-mode routine entrypoint          |   2 S$EPT
|---------------------------------------------------|
| CSM  (PC)+                                        |   4 S$CSM
|---------------------------------------------------|
| Address of Completion routine in library          |   6 S$CRA
+---------------------------------------------------+
```

Library Updates

revision
PRO-123

BASIC, PASCAL, PortaCalc, Kermit and a Desk Top Calendar

Version: V3, October 1984

Author: Various

Submitted By: Glenn C. Everhart, Ph.D.

Operating System: P/OS, RSX-11M-PLUS

Source Language: C, FORTRAN 77, MACRO-11

Other Software Required: PASCAL is not useful without the PRO Toolkit

These diskettes contain a grab bag of several RSX-11 tools off old RSX-11 SIG tapes, converted to RX50 format for the DEC Professional-350 under P/OS V1.7 or later (Maybe earlier too; no way to test that).

The following are provided:

PortaCalc for Professional-350 is the most powerful spreadsheet available. This version is compatible with the VAX/VMS version and does essentially everything you ever wanted your spreadsheet to do. The tutorial and manual file are supplied, and an install file is there too, in case you want to run it from a menu.

DTC for Professional-350 is a good desktop calendar facility for handling your calendar on the PRO. Full screen day/week/month/year displays, appointment selection, meeting scheduling, etc. etc.

RSX BASIC for the Professional-350 is the Michael Reese BASIC, a full language dialect similar to BASIC PLUS.

SWEDISH PASCAL for the Professional-350 is the latest "Swedish PASCAL" compiler (updated for the new RSX versions) for the PRO. It is a full language compiler and OTS, plus the manual. This program is useful for those who have the PRO Toolkit so they can build programs in PASCAL now.

PRO Kermit V1 is a full featured communications package with host versions in the public domain available for practically any host or other Micro you ever heard of, and many you haven't. This package allows full VT102 emulation, file transfer, logging and much more and makes it totally unnecessary for you to buy anybody's communications packages. Includes the hexify and dehexify tools, making it possible to send even files with weird rms attributes around and rebuild the attributes.

There are install command files for BASIC, DTC, PortaCalc, and Kermit. The PASCAL kit is to be run from the PRO Toolkit, so is not installable from a menu. The others can run under the PRO Toolkit too; it is not necessary to go through the menu, except maybe for Kermit, which is heavily into PRO menus.

Note: Complete documentation and sources were not included to reduce the number of floppies. Documentation and sources for most of the programs in this package can be found with DECUS Program Numbers: 11-SP-18, 11-SP-47, 11-SP-60, and 11-SP-67.

Changes and Improvements: Cleaned up speedups for PortaCalc BASIC. Added a couple of C tools.

Complete sources not included. Documentation on magnetic media.

Media (Service Charge Code): 5 1/4" Floppy Diskette (JD)

Format: FILES-11

Keywords: Spreadsheet, Calendars, Compilers, PASCAL, BASIC, Kermit
Operating System Index: RSX-11/IAS, P/OS

November 26, 1984

new
11-765

RSX SIG Tapes Evaluation

Version: October 1984

Author: A. Szentgali

Submitted By: Klaus Centmayer, TU Muenchen, Munich, West Germany

Operating System: IAS, RSX-11M

This collection of reports is a review of programs from the DECUS RSX Symposium Tapes. Its goal is to evaluate the programs and their building procedures and to help users in choosing and installing software according to their actual needs and configuration. Testing includes building and installation procedure and, as far as possible, a brief run test. This report contains the US-RSX-SIG-Tapes Spring and Fall 82 (up to now not complete).

The tape includes a SIG-Tape Road Map Summary as a quick reference. It contains:

RSX-IAS US Fall'77...Fall'83, Europe'79...'83
PASCAL Spring'80...Fall'81, RT-11 Fall'79...Fall'81
Lars Palmer + IAS-ICR collections.

This tape contains documentation only.

Media (Service Charge Code): 600' Magtape (MA)

Format: FILES-11

Keywords: Symposia Tapes -
RSX-11
Operating System Index:
RSX-11/IAS

November 26, 1984

revision
V-SP-28

Symposium Tape from the RSX SIG, Spring 1984, Cincinnati, in VMS/BACKUP

Version: V2, Spring 1984

Author: Various

Submitted By: Glenn C. Everhart, Ph.D.

Operating System: IAS, RSX-11D, RSX-11M, RSX-11M-PLUS, VAX/VMS

Source Language: Various

This program is the RSX, Spring 1984, Cincinnati Symposium tape for the convenience of VMS users. It is available in either BRU format (DECUS Program No. 11-SP-67) or VMS/BACKUP format at 1600 BPI.

The following are some highlights of the tape:

Michael Reese BASIC, ATT, DEV, Force command line, DOS cross supports, virtual disks (memory and disk resident), spreadsheet, calendar, Runoff from Rice University, spelling checker, several communications utilities including updates and extensions of XMITR and DUPLEX, graphics, full Kermit distribution (as of 7/15/1984), BUG screen debugger for IAS, LBL Tools toys (LEX, YACC, LISP, RATFOR, AR, etc.), DECUS C for PRO-350, phone list manager, mailing list manager, sorter, HEX file mgr, Task Image Zap, and numerous PRO-350 loadable task images and runtimes including Swedish PASCAL, BASIC, DTC, AnalytiCalc (spreadsheet), DDT (sources for symbolic debugger), TECO SRD, COPY, LISTRSX, and much more. Since many of these programs are as useful on VAX as on RSX, they are provided.

No guarantees are made as to the completeness, usability, or quality of the programs on the tape. The material has not been checked or reviewed and documentation may or may not be included.

Changes and/or Improvements: Some new Kermits and a significant PortaCalc speedup.

Complete sources not included. Documentation on magnetic media.

Media (Service Charge Code): 2400' Magtape (PS)

Format: VMS/BACKUP (Blocked at 32768)

Keywords: Symposia Tapes - VMS, Spreadsheet, BASIC, RUNOFF
Operating System Index: RSX/IAS, VAX/VMS

November 26, 1984

C Language System with Native Toolkit

Version: November 1983

Author: David Conroy, Robert Denny, Charles Forsythe, Clifford Geshke and Martin Minow

Submitted By: Martin Minow

Operating System: P/OS V1.8

Source Language: C, MACRO-11

"C" is a general purpose programming language well suited for professional usage. This "C" distribution contains a subset of the DECUS "C" programming system which includes:

- A compiler for the "C" language. The entire language is supported except for an emulated (software) floating point, macros with arguments, bit fields, and enumerations.
- A common runtime library ('standard I/O library') for "C" programs running under the RSX-11 or RT-11 operating systems. By using this library, "C" programs may be developed on one operating system for eventual use on another.
- An RSX-11/M extensions library allowing access to all RSX-11M executive services.

Note: For sources and documentation manuals see DECUS No. 11-SP-18.

Restrictions: This submission contains neither documentation nor sources for the compiler or Run-Time Library. The "tools" are not provided. Some functions for accessing P/OS menus are provided.

Sources not included. Very limited documentation on magnetic media.

Media (Service Charge Code): 5 1/4" Floppy Diskette (JC)

Format: FILES-11

Keywords: Programming Languages
Operating System Index: PO/S

November 12, 1984

revision
11-528

TAB: A Low-Overhead Data Management System for the PDP-11

Version: April 1984

Author: R.N. Stillwell, Baylor College of Medicine, Houston, TX

Operating System: IAS V3.1, RSX-11M V2.1

Source Language: MACRO-11, FLECS

Memory Required: 28KW

This package builds a set of tasks providing a small, relatively unsophisticated data management system in which the user can easily define and manipulate tables of data. Tables are arrays of rows and columns containing data in character form. Tasks are provided to define a table, modify the format of an existing table, update and list the contents of a table, and split and merge tables. An interpreter for a simple language (RPT) provides a means of writing specific applications, including report generation and table updating: it has provision for arithmetic and string operations, terminal interaction, file I/O, conditional and repetitive execution, and subtasking. The 1984 release provides a full-screen table editor on VT100-compatible terminals, and all necessary modifications for FORTRAN-77 and RSX11M/M+ (including subtasking). Command and batch files for building and testing the package, and sources for the Flecs preprocessor are included. A spreadsheet application written in RPT is provided as an example.

Documentation on magnetic media.

Media (Service Charge Code): 600' Magtape (MA)

Format: DOS-11

Keywords: Data Base Management
Operating System Index: RSX-11/IAS

November 12, 1984

new
11-760

COMPOSE: VT200 Custom Character Set Generator Program

Version: V1.0, October 1984

Author: Bob Awde, Jr., General Mills, Minneapolis, MN

Operating System: RSX-11M V4.1, RSX-11M-PLUS V2.1

Source Language: FORTRAN 77, MACRO-11

Memory Required: 15,264 Words

Special Hardware Required: VT200 Family of Terminals

The COMPOSE program permits you to design and automatically generate custom character sets for the VT200 family of terminals. The output of COMPOSE consists of two files; a FORTRAN direct access file that contains the character definitions in binary form and a test file that can be "typed" at an appropriately configured VT200 terminal to actually create the custom character set. An example set of files used to generate the APL character set is included.

Documentation on magnetic media.

Media (Service Charge Code): Write-Up (AA), Floppy Diskette (KA), 600' Magtape (MA)

Format: FILES-11

Keywords: Utilities - RSX-11, Terminal Management
Operating System Index: RSX-11/IAS

November 12, 1984

new
11-SP-72

Reese BASIC

Version: May 1984

Author: Frank Borger, Michael Reese Hospital, Chicago, IL

Operating System: IAS V3.01 and later, RSX-11M V3.2 and later

Source Language: MACRO-11

Memory Required: Various

Reese BASIC is a highly upgraded version of what used to be a DECUS library program for DOS.

1. Full FILES-11 I/O is supported, (fixed length random access, shared mode, etc.).
2. String functions and user defined functions are much more flexible than in either the original version or in DEC's BASIC-11.
3. Multi-user implementation is supported with separate pure and impure areas (IAS and RSX-11D only).
4. Since it is an interpreter, it includes the special debugging commands: STEP, CON and SET TRACE.
5. Although an interpreter, significant manipulation of the source program is done to speed up operation.
6. OVERLAY and a data preserving CHAIN are also supported.
7. A clean "break" feature is implemented via the TT handler.
8. A number of BASIC-PLUS2-like features have been added including: virtual arrays, integer and byte variables, continued lines and IF-THEN-ELSE.
9. User written machine language subroutines are supported.
10. The capability of SPAWNING another task is supported.

Also included on the tape are two entertainment programs, MURPHY, which randomly produces versions of Murphy's law, and MAY, which gives curses of the form "May the bird of happiness..."

Documentation on magnetic media.

Media (Service Charge Code): 600' Magtape (MC)

Format: DOS-11

Keywords: Programming Languages, BASIC
Operating System Index: RSX-11/IAS

October 29, 1984

DECUS

DECUS SUBSCRIPTION SERVICE
DIGITAL EQUIPMENT COMPUTER SOCIETY
249 NORTHBORO ROAD, (BPO2)
MARLBORO, MA 01752

MOVING OR REPLACING A DELEGATE?

Please notify us immediately to guarantee continuing receipt of DECUS literature. Allow up to six weeks for change to take effect.

( ) Change of Address
( ) Delegate Replacement

DECUS Membership No.: ____________
Name: ____________
Company: ____________
Address: ____________
____
State/Country: ____________
Zip/Postal Code: ____________
Phone No.: ____________

Mail to: **DECUS - Attn: Subscription Service**
249 Northboro Road, (BPO2)
Marlboro, MA 01752 USA

[redacted]

Bulk Rate
U.S. Postage
PAID
Permit No. 18
Leominster, MA
01453