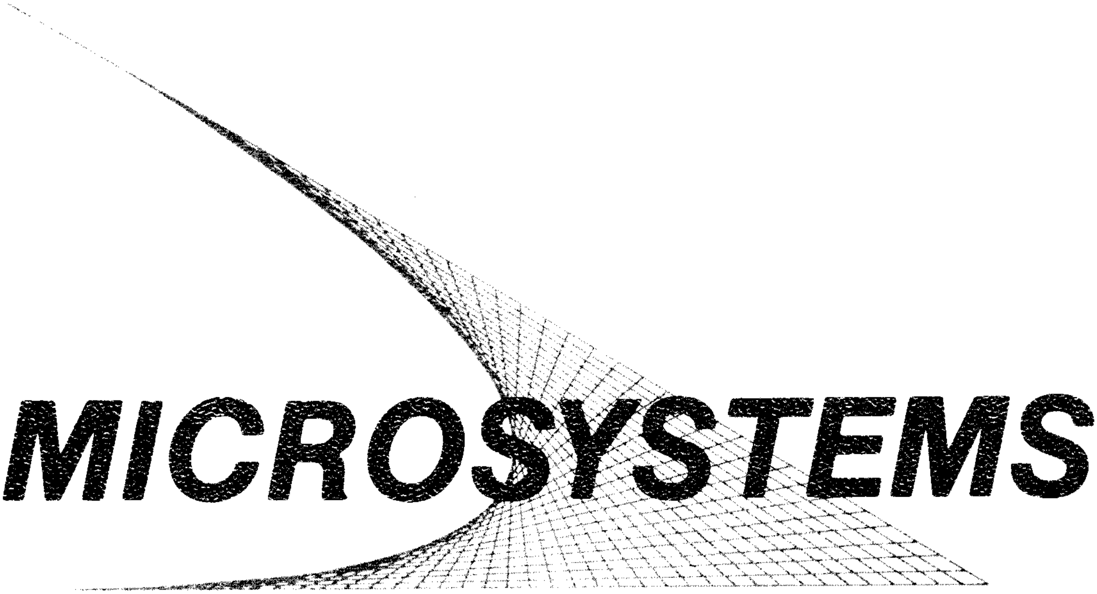
MOTOROLA

M68KRMSTI/D2

RMS68K/VERSAdos
Table-Driven Task Initiator
Reference Manual



QUALITY • PEOPLE • PERFORMANCE

MOTOROLA

M68KRMSTI/D2

DECEMBER 1985

RMS68K/VERSAdos

TABLE-DRIVEN TASK INITIATOR

REFERENCE MANUAL

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, Motorola reserves the right to make changes to any products herein to improve reliability, function, or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights or the rights of others.

Second Edition

First Edition October 1985

MICROSYSTEMS

REVISION RECORD

M68KRMSTI/D2 -- December 1985. Incorporates information contained in both M68KRMSTI/D1 and S1, and adds a keyword index.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

1.1 GENERAL

This manual describes the Table-Driven Task Initiator (TDTI) operating under VERSAdos or RMS68K. It describes what a TDTI system is, describes what it can do, and provides detailed instructions in the use of such a system. A comprehensive example and instructions for generating an example system using furnished chainfiles are included in this document.

1.1.1 TDTI Software

A TDTI system consists of two parts. The first part, which is supplied in the software package, consists of the following:

- RMS68K kernel
- Standard RMS68K system initializer
- Procedure portion of the TDTI
- Optional I/O subsystem which consists of user-selected drivers and the VERSAdos I/O subsystem tasks File Handling Services (FHS) and Input/Output Services (IOS) (these items are not included for an RMS68K-type system)

The second part of the system consists of user-written code:

- All user-written application tasks
- All user-written non-standard startup code and error handling logic
- The Task Table (TT) that TDTI will process

1.1.2 Operating Environment

The Task Initiator is run, in association with RMS68K, on a target machine in an environment where application software is developed on a host machine and downloaded to be tested on a target machine. The host machine can be any M68000-family system running VERSAdos. The target machine can be any M68000-family system with RMS68K.

1.2 FUNCTIONAL DESCRIPTION

1.2.1 Purpose

A TDTI system is an RMS68K or VERSAdos operating system along with some user-written tasks and some "special" code called TDTI which is used to start the other tasks in the system in a controlled way. A TDTI system provides the following capabilities:

a. Assists in debug effort by giving the user complete control over start-up processing in the system. If a system consists of several tasks, the user can elect to start these tasks in any order by making simple changes to a table. If the system consists of a number of tasks but the user is concerned only with a small subset, it is possible to prevent some tasks from starting at all by making minor table modifications.

b. Saves time by allowing the user to make modifications to the system without having to initiate a SYSGEN process every time. If the user desires to replace a task, the module to be replaced and the SYSGENed table-driven system can be downloaded, the Program Counter (PC) can be set, and processing can continue. Thus, the user can easily replace a task and eliminate the need for having to re-SYSGEN.

c. Allows the user to define code to handle errors during startup processing. The error handling code can be as simple as a branch-to-self which will "hang" the system and allow for postmortem debug analysis, or extremely sophisticated code which may attempt error recovery.

d. Provides an easy path to ROMing the end product. The RMS68K, TDTI, and most VERSAdos components are directly ROMable. Once the system is debugged in RAM, ROMing is a trivial step. Note that the VERSAdos file management subsystem, Session Control, and the loader are not ROMable, but the I/O subsystem and drivers are.

The user is not restricted to the predefined "standard" functions that the TDTI program performs. If "non-standard" processing to startup a particular task is desired, the user may write code to be executed in lieu of the "standard" processing that TDTI would normally perform for that task, while still using the TDTI "standard" functions for other tasks in the same system.

TDTI performs several functions on behalf of a task during its "standard" processing. Some examples are: "create Task Control Block (TCB)", "task segment allocation", and "setting the task initial state to dormant or executing". All of these functions are directly accessible to user code as subroutines. Thus, if the user desires to write some special startup processing for a particular task, some of these "standard" functions may be found to be useful and thus less special purpose code would have to be written.

1.2.2 Operation

A typical TDTI system can be started in one of several ways:

a. The system can be booted into RAM from disk and given control from the Initial Program Loader (IPL).

b. The system can be downloaded from a host using debug firmware commands and manually started, by setting the PC and the stack register A7 and entering the GO command.

c. The system may be embedded; that is, it may exist entirely in ROM and be given control during a power-up sequence.

When the TDTI system has control, the following functions are performed:

a. The first process to get control is the system initializer. It is really an extensive initialization process that sets up the proper environment for RMS68K to run. Some of the functions it performs are:

   . Initializes the system vectors.

   . Initializes memory and builds free memory lists.

   . Initializes System Parameter Area (SYSPAR).

   . Initializes the Memory Management Unit (MMU) if one exists.

   . Initializes any special devices such as timers or special I/O devices.

   . Sets up system tables for:

ASN	Address segment numbers
GST	Global segment table
UST	User semaphore table
IOV	I/O vector map
PAT	Periodic activation table
UDR	User-defined directive table

   Most of the functions performed by the system initializer are controlled by parameters defined at system generation (SYSGEN) time. Do not confuse these functions with the startup functions performed by TDTI. The functions performed by the TDTI are controlled by a table that the user can easily modify without initiating a new SYSGEN process. For a complete discussion of SYSGEN-related parameters, refer to the System Generation Facility User's Manual.

b. After the system initializer has completed its initialization, control is passed to the RMS68K kernel. The dispatch sequence initiated by the kernel is as follows:

For an RMS68K TDTI system. The kernel will dispatch the TDTI task, as it is the only task that has been defined to the system. The user-written tasks will not be defined to the system until TDTI has executed.

For a VERSAdos TDTI system. The kernel will dispatch the I/O subsystem task. Prior to termination, the I/O subsystem task will ensure that the TRAP #2 and #3 I/O handlers are ready for execution; then it will start the TDTI task, which in turn will make the user-written task known to the system.

c. The function of the TDTI task is not complex. It operates on a user-written table that can contain any number of entries. The entries are processed in an order that is determined by a "priority" field that is part of the task entry itself. Normally an entry will contain the information that the TDTI needs in order to define a task to RMS68K. An entry such as this will be referred to as a "standard" entry and the processing associated with such an entry will be termed "standard" processing. Standard processing consists of the following:

1. Creating a TCB.
2. Allocating segment(s) to the task associated with the entry.
3. Optionally starting the task.

Most of the information in a single table entry is the information required to perform the above functions. Some additional information is required in the table, however. This additional information falls into the following groupings:

1. Control-related information consisting of the following:

   . Processing priority which defines the order in which entries in the table are processed.

   . Links which point table entries to successive entries and links which point to non-standard user defined code.

2. Debugging aids consisting of user defined "eye catchers".

A hierarchical relationship of RMS68K, the TDTI, the TT, and the associated task is shown in Figure 1-1.

1.3 CONVENTIONS USED IN THIS MANUAL

The following conventions are used in the command syntax, examples, and text in this manual:

boldface strings	A boldface string is a literal such as a command or a program name, and is to be typed just as it appears.
italic strings	An italic string is a "syntactic variable" and is to be replaced by one of a class of items it represents.

Operator inputs are to be followed by a carriage return. The carriage return is shown as (CR) only if it is the only input required.

1.4 RELATED DOCUMENTATION

The following publications may provide additional helpful information. If not shipped with this product, they may be obtained from Motorola's Literature Distribution Center, 616 West 24th Street, Tempe, AZ 85282; telephone (602) 994-6561.

DOCUMENT TITLE	MOTOROLA PUBLICATION NUMBER
System Generation Facility User's Manual	M68KSYSGEN
M68000 Family Real-Time Multitasking Software User's Manual	M68KRMS68K
VERSAdos Data Management Services and Program Loader User's Manual	RMS68KIO
M68000 Family Linkage Editor User's Manual	M68KLINK
M68000 16/32-Bit Microprocessor Programmer's Reference Manual	M68000UM

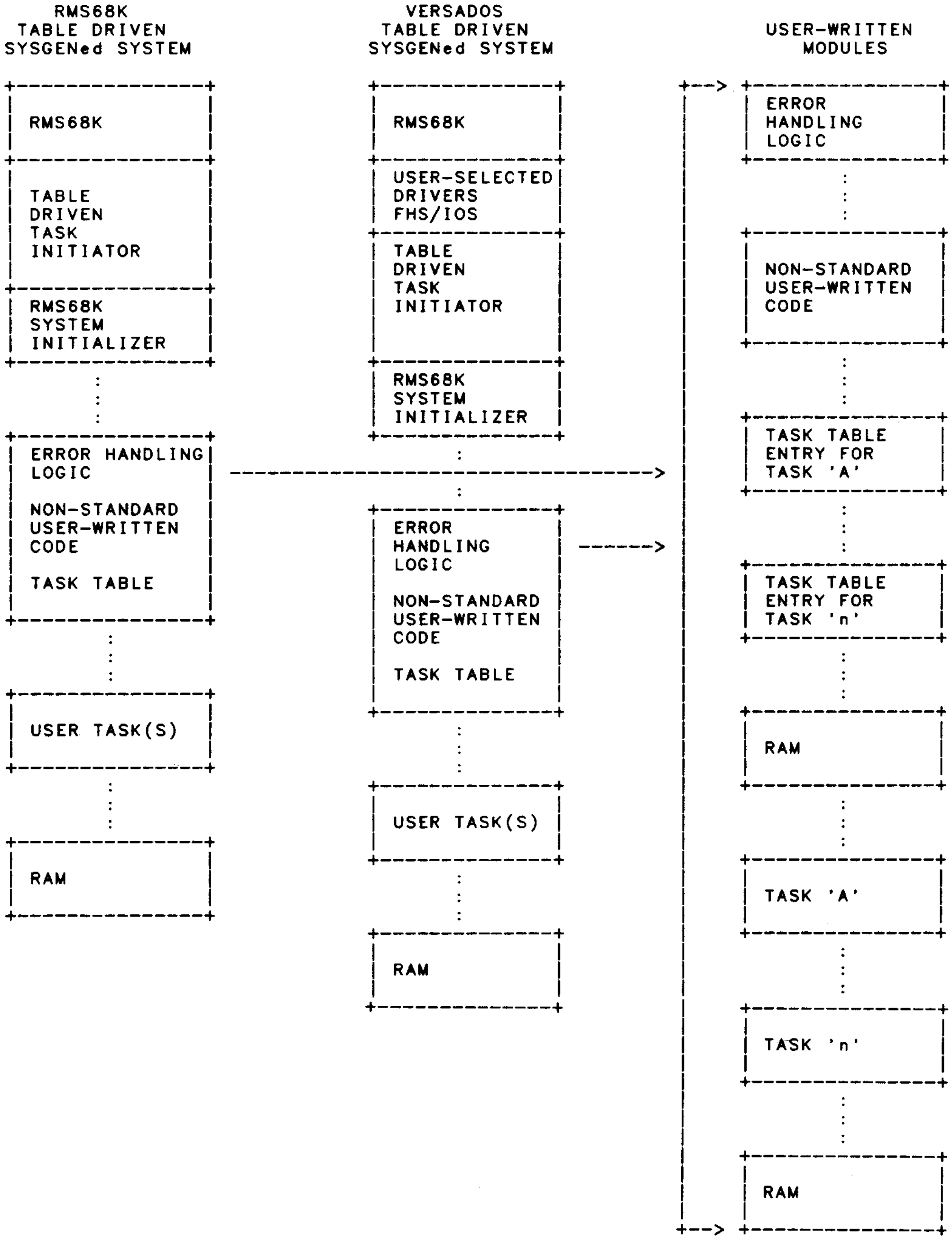

RMS68K TABLE DRIVEN SYSGENed SYSTEM
RMS68K
TABLE DRIVEN TASK INITIATOR
RMS68K SYSTEM INITIALIZER
ERROR HANDLING LOGIC
NON-STANDARD USER-WRITTEN CODE
TASK TABLE
USER TASK(S)
RAM
VERSADOS TABLE DRIVEN SYSGENed SYSTEM
RMS68K
USER-SELECTED DRIVERS FHS/IOS
TABLE DRIVEN TASK INITIATOR
RMS68K SYSTEM INITIALIZER
ERROR HANDLING LOGIC
NON-STANDARD USER-WRITTEN CODE
TASK TABLE
USER TASK(S)
RAM
USER-WRITTEN MODULES
ERROR HANDLING LOGIC
NON-STANDARD USER-WRITTEN CODE
TASK TABLE ENTRY FOR TASK 'A'
TASK TABLE ENTRY FOR TASK 'n'
RAM
TASK 'A'
TASK 'n'
RAM

FIGURE 1-1. Hierarchical Relationships Within TDTI System

CHAPTER 2

TASK TABLE

2.1 GENERAL

Understanding how to use the TDTI system requires a thorough understanding of the Task Table (TT) format. The table provides a flexible way for the user to control the behavior of the system at startup time.

The TT consists of a header followed by one or more Task Table Entries (TTE). (See Figure 2-1.) A utility program, TTGEN, is furnished to simplify creation of a TT. It is described in paragraph 3. Refer also to Appendix A for a sample TT.

The header has a self-relative link to the first TTE. Each TTE in turn is linked to the next TTE by a self-relative offset. The last TTE contains a self-relative offset value of zero.

```
        Task Table Format             Task Table Entry Format

     +------------------------+     +------------------------+
     |                        |  +->|                        |
     | Task Table             |  |  | Task Entry 1           |
  +--| Header Information     |  |  | Header Information     |
  |  |                        |  |  |                        |
  |  +------------------------+  |  +------------------------+
  |  |                        |  |  |                        |
  +->| Task Table Entry  1    |-->|  | Task Entry 1           |
     |                        |  |  | Task Control Block     |
  +--|                        |  |  | Related Information    |
  |  +------------------------+  |  |                        |
  |->           .                |  +------------------------+
                .                |  |                        |
                .                |  | Task Entry 1           |
  +--           .                |  | Segment 1 Allocation   |
  |  +------------------------+  |  | Related Information    |
  |  |                        |  |  |                        |
  +->| Task Table Entry  n    |  |  +------------------------+
     | (Self-relative offset  |  |              .
     |  equals zero)          |  |              .
     +------------------------+  |              .
                                 |              .
                                 |  +------------------------+
                                 |  |                        |
                                 |  | Task Entry 1           |
                                 |  | Segment n Allocation   |
                                 |  | Related Information    |
                                 +->|                        |
                                    +------------------------+
```

FIGURE 2-1. Task Table Entries

Each TTE contains information required to make the task known to the operating system by creating the Task Control Block (TCB), allocating the segment(s) required by the task, and setting the initial state of the task to dormant or executing.

The TT and all of the TTEs typically would appear as a contiguous block of data in RAM. However, this is not a requirement since the task table is a linked structure.

2.2 SELF-RELATIVE LINKS

Any pointers or links used in the TT are "self-relative". This means that the link is a 4-byte field containing the PC-relative offset from the link to the data or code item. As an example, consider the following:

Assume that A0 contains the absolute address of a data structure ($1000) and the contents at offset LINK ($500) in this data structure consists of the PC-relative offset ($200) to data item B. Then data item B would be located at physical address $1700.

One way to obtain the address of data item B in A4 would be to execute the following sequence of code:

```
LEA      LINK(A0),A4       A4  contains the address  of the
                           link

ADD.L    (A4),A4           A4 contains the absolute address
                           of data item B which is equal to
                           the sum  of the absolute address
                           of  LINK  plus  the contents  of
                           LINK
```

```
                                                ABSOLUTE ADDRESS

A0 ---------> +--------------------+          $1000
              |                    |
              |                    |
              |                    |
              |                    |
              |                    |
              +--------------------+          $1500
LINK ($500)   | Offset to data     |
              |  item B ($200)     |
              +--------------------+
              |                    |
              |                    |
              |                    |
              |                    |
A4 ---------> +--------------------+          $1700
              | (Data item B)      |
              |                    |
              +--------------------+
```

2.3 TASK TABLE FIELD DEFINITIONS

The following illustrations describe the field contents of the TT. Shown with each description is the mnemonic that provides offset identification, used by TDTI, associated with the field. Explanations of the fields follow the illustrations and have been grouped by function. The letter following each mnemonic identifies the functional grouping where this field is described in detail.

Task Table Header

TTSUHID	(A)	User-generated table header identifier
TTSEHLOS	(B)	Self-relative offset to error handling logic
TTSFTEOS	(B)	Self-relative offset to first task entry

Task Table Entry

TTSUTEID	(A)	User-generated task entry identifier
TTSNEOS	(D)	Self-relative offset to next task table entry
TTSUSC	(D)	Self-relative offset to non-standard code
TTSTSKPO	(C)	Task processing order
TTSINITS	(G)	Task initial state
TTSTN	(E)	Taskname
TTSTSN	(E)	Task session number
TTSTCBDO	(E)	TCB directive options
TTSMTN	(E)	Monitor taskname
TTSMTSN	(E)	Monitor task session number
TTSTIP	(E)	Task initial priority
TTSTLP	(E)	Task limit priority
TTSTCBTA	(E)	TCB task attributes
TTSTEPA	(E)	Task entry point address
TTSUGID	(E)	Task user-generated identification
TTSNOS	(F)	Number of segments

Task Table Entry Segment Information

```
                    +----------------------------------------------------+
TTSGSDO     (F)     | Segment directive options                          |
                    +----------------------------------------------------+
TTSGSSA     (F)     | Segment attributes                                 |
                    +----------------------------------------------------+
TTSSEGNM    (F)     | Segment name                                       |
                    +----------------------------------------------------+
TTSSEGAD    (F)     | Segment address                                    |
                    +----------------------------------------------------+
TTSSEGSZ    (F)     | Segment size in bytes                              |
                    +----------------------------------------------------+
```

a. Debug Aids

There is no function associated with these fields other than to allow the user to identify easily the start of the entity described. Typically these identifiers will be a string of up to four ASCII characters that would easily be recognized in a memory dump.

TTSUHID: 4 bytes -- User header identifier

This field is a user-generated TT header identifier, i.e. "!HDR".

TTSUTEID: 4 bytes -- User task entry identifier

This field is a user-generated task entry identifier, i.e. "TE01".

b. Header Related Fields

TTSEHLOS: 4 bytes -- Error handling logic offset

This field contains a self-relative offset to the user-written error handling logic to be executed if the TDTI task encounters an error while processing the TT. A simple error handling routine that would "hang" the system and allow postmortem debug is a Branch-to-Self instruction. Upon entering the error handling logic register, A4 will point to the TTE being processed, A5 will point to the TT header, and A7 will point to an Error Index Value. (Refer to Appendix E.)

The user may elect to bypass the error handling capability in one of two ways:

1. By placing a value of zero in this field.

2. By setting bit 0 in this field, giving an odd self-relative offset.

If either of the above is done and an error is encountered while processing a TTE, processing on that entry will terminate and execution will commence with the next TTE. Bit 0 was selected because the user could easily set and reset this bit without destroying the even value of the self-relative offset.

If the user elects to return from the error handling logic, via an RTS instruction, he must preserve the stack pointer. Preservation of all other registers will be done by the TDTI task. Upon return, processing will commence with the next TTE.

TTSFTEOS: 4 bytes -- First task entry offset

This field contains the self-relative offset to the first TTE in the TT. A value of zero implies that there are no TTEs.

c. Priority Processing Order Value

TTSTSKPO: 2 bytes -- Task priority processing order value

This field contains the task priority processing order value. The execution order is from low to high. Assume the TT has four TTEs as follows:

TT ENTRY	PROCESSING ORDER VALUE
1	31
2	10
3	17
4	35

The TTE processing order would be TTE2, TTE3, TTE1, TTE4.

If multiple TTEs with the same processing order value exist, the TTEs will be processed sequentially from the first entry to the last entry.

Assume a TT had four TTEs as follows:

TT ENTRY	PROCESSING ORDER VALUE
1	31
2	05
3	17
4	05

The TTE processing order would be TTE2, TTE4, TTE3, TTE1.

The user can bypass processing of a TTE by setting bit 15 in this field. Bit 15 was selected because the processing order value will never be negative and the user can easily set and reset this bit without destroying the processing order value.

The user may elect to disregard this field entirely, leaving a value of zero in each TTE, in which case the entries will be processed in the exact order as they are defined in the table.

d. Links Not Associated with the TT Header

TTSNEOS: 4 bytes -- Self-relative offset to the next table entry

This field contains the self-relative offset to the next TTE in the TT. A value of zero implies there are no more TTEs in the TT.

TTSUSC: 4 bytes -- Self-relative offset to non-standard user code

The user can elect to have TDTI perform the "standard" processing functions for a particular TTE in the TT, or he can elect to assume these responsibilities with his own code. Typically this would be done if a task required unique or additional functions which are not a part of TDTI's "standard"' processing. If the user elects to assume this responsibility, this field contains the self-relative offset to the non-standard user code to be processed for this task.

If non-standard code is to be executed, a Jump-to-Subroutine (JSR) call will be made to the non-standard code. The user is responsible for maintaining the stack pointer and returning via an RTS instruction. Preservation of all other registers will be done by the TDTI task. On a call to the non-standard code, register A5 will point to the TT header, and register A4 will point to the TTE to be processed. Routines normally executed by the TDTI task will be accessible to the non-standard user code as subroutines, via a JSR. These routines include:

. Character-fill a work area

. Create a TCB for a task

. Call the task table error handling logic, which ultimately executes the user-written error handling code

- Obtain the task identification, which is required usage for a realtime task
- Allocate segments for a task
- Start a task

Refer to Appendix D for details regarding these subroutines.

Upon return from the non-standard user code, the TDTI task commences processing with the next TTE in the TT.

Since all non-standard user-written code that is executed operates as a subroutine to TDTI, the user can execute any RMS68K directive desired. For example, it is possible that the user may wish to execute a delay request as a coordination effort before moving on to the next entry in the TT.

An example of this is demonstrated with entry number 5 in the module 9990..TTS.SA and the corresponding user code in module 9990.&.TE05USC.SA. (Refer to Appendix B.)

Note that in this example, which has non-standard user code, only the following fields were defined:

TTSNEOS -- Self-relative offset to next TTE

TTSUSC -- Self-relative offset to non-standard user code

TTSTSKPO -- Task priority processing order value

The user may elect not to execute existing non-standard code by setting bit zero of this field. In this case, the TTE will be processed as if the non-standard code did not exist. Bit zero was selected because the user could easily set and reset this bit without destroying the even value of the self-relative offset.

A value of zero implies there is no non-standard user code.

e. Create Task Control Block Information

This information is related to the RMS68K directive "CRTCB" found in the M68000 Family Real-Time Multitasking Software User's Manual.

TTSTN: 4 bytes -- Taskname

This field contains the taskname.

TTSTSN: 4 bytes -- Task session number

This field contains the session number of the task.

TTSTCBDO: 2 bytes -- TCB directive options

This field contains the TCB directive options.

TTSMTN: 4 bytes -- Monitor taskname

This field contains the monitor taskname for this task entry.

TTSMTSN: 4 bytes -- Monitor task session number

This field contains the session number of the monitor task.

TTSTIP: 1 byte -- Initial priority

This field contains the task initial priority.

TTSTLP: 1 byte -- Limit priority

This field contains the task limit priority.

TTSTCBTA: 2 bytes -- TCB task attribute

This field contains the TCB task attributes.

TTSTEPA: 4 bytes -- Entry point address

This field contains the task entry point address.

TTSTUGID: 2 bytes -- Task ID

This field contains a user-generated task identification.

f. Segment Handling Information

TTSNOS: 2 bytes -- Number of segments

This field contains the number of segments, range of one to four, which TDTI will allocate to this task.

The following information is related to the RMS68K directives "GTSEG", "DCLSHR", and "SHRSEG" found in the M68000 Family Real-Time Multitasking Software User's Manual.

The segment directive options and the segment attributes reflect a subset of options and attributes that have been extracted from the TRAP #1 calls of GTSEG, DCLSHR, and SHRSEG. This subset has been determined to give the maximum flexibility for the following logic sequence which the TDTI task processes.

If a segment is declared shareable, an attempt will be made to grant shared access for the requesting task. If shared access is denied, TDTI will allocate the segment to itself, declare it shareable, and then transfer it to the target task. TDTI will perform segment allocation for the target task if shared access is not requested.

The appropriate directive options and segment attributes will be selected for the TRAP call being initiated.

The description of the last segment associated with the task will complete the definition of the TTE.

TTSGSDO: 2 bytes -- Segment directive options

Bit	Meaning
13	0 - An address is specified in the address field for this segment. 1 - RMS68K supplies logical address equals physical address.
12	0 - Shareable segment is not permanent 1 - Shareable segment is permanent.
08	0 - RMS68K does not attempt to allocate the segment at the physical address specified in the address field for this segment. 1 - RMS68K attempts to allocate the segment at the physical address specified in the address field for this segment. Since this is a physical address, the logical address will equal the physical address.

TTSGSSA: 2 bytes -- Segment attributes

Bit	Meaning
14	0 - Segment is to be read/write. 1 - Segment is to be read only.
13	0 - Segment is not locally shareable. 1 - Segment is locally shareable.
12	0 - Segment is not globally shareable. 1 - Segment is globally shareable.
11	0 - Segment is not memory mapped I/O space. 1 - Segment is memory mapped I/O space. The address given in the address field for this segment must be a physical address that is not in the limits of allocatable RAM. If this bit is set, none of the GTSEG options are applicable and the segment is allocated as as a shared segment.

TTSSEGNM: 4 bytes -- Segment name

This field contains the segment name.

TTSSEGAD: 4 bytes -- Segment address

This field contains the segment address.

TTSSEGSZ: 4 bytes -- Segment size

This field contains the segment size.

g. Task Initial State Information

TTSINITS: 2 bytes -- Task initial state

This field contains the code of the initial state that this task is to have. Valid codes are "R" (ready to execute) and "D" (this task is dormant).

If the initial state does not have a valid value, it will default to dormant, "D".

The high order byte of this word has been reserved for future use:

```
+----------+----------+
| Future   | State    |
| Use      | Code     |
+----------+----------+
```

2.4 CREATING THE TASK TABLE

TDTI requires a complex TT to drive its startup processing. The complex table format provides a great deal of flexibility, but is difficult to create manually. The Task Table Generator Utility (TTGEN) provides an easy way for the user to create this table. The utility provides a series of simple menus which allow creation of a TT of arbitrary length. Most of the low level details in the table creation are hidden, thus minimizing the amount of information with which the user must deal.

The output from TTGEN is assembly language source which has been commented to make various parts of the table easy to find. After the output has been created, it can be combined with other user-written source code and assembled.

TTGEN will query the user for information required to build the table. Some of the information required are the names of the load module files representing the tasks that the user wishes to integrate into the system. The load module files named must be available for TTGEN to access. Some of the information that TTGEN puts into the table is obtained from the first sector of the load module file (known as the Loader Information Block -- LIB).

Because the user interface is very simple, TTGEN may not be able to generate some of the more exotic options available through use of the TT. It is possible, however, to generate a table that is very nearly complete with TTGEN. Any options that cannot be handled by TTGEN can easily be inserted into the table with the VERSAdos editor.

2.4.1 Invoking TTGEN

The command line to invoke TTGEN is the following:

=TTGEN [*output field*]

where:

output field is the filename which will receive the output from TTGEN.

If *output field* is not entered, the output filename will default to TTABLE.SA under the current default volume, catalog, and user number. The output field must be a file; it cannot be a device such as #PR.

An initial selection menu will be presented to the user. There are three "work" menus and three "help" menus. Each help menu is associated with a single work menu. Movement between menus can only take place along the lines indicated.

The following diagram shows all of the menus and their relationship to one another:

```
                        Initial selection menu
                              |    |    |
                              |    |    |
            ------------------     |   ------------------
            |                      |                    |
 Define task table header          |                    |
            |                      |  Help menu for initial selection
 Help menu for header              |
                                   |
                      Define single task entry
                                   |
                      Help menu for task entry
```

After invoking the TTGEN utility, TTGEN will determine if the output file already exists. If so, TTGEN will present the following question to the user.

Output file "*filename*" exists - OK to overwrite? (Y/N)

where *filename* is the completely qualified output filename. If the user answers with **N**, the utility will terminate with no further action taken. If the user answers with **Y**, the utility will continue. The output file will not be overwritten, however, until the user gives the command to configure (build) the TT.

2.4.2 "Initial Selection" Menu

The "initial selection" menu, along with its associated help menu, is shown below:

```
--------------------------------------------------------------------------

Main selection:

   1  -  Define Task Table Header

   2  -  Define or modify a single table entry

   3  -  Delete a single table entry

   C  -  Configure (Build) the Task Table

   Q  -  Quit

   H  -  Help

Enter selection >
--------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------

                    H E L P (for Initial Selection Menu)

 Selection   Description

 1  -  (Optional)  Displays the menu which allows you to define the task table
       header.  If  not  selected, the task table header will be defined with
       default values.

 2  -  Displays the menu which allows you to define an entry in the task table
       for  a  single  task.   A  load module file for the task must have been
       created prior to building the table entry for that task.

 C  -  Builds  the  task table output file using current task table header and
       task  table  entry  values.   If  an  output  file  exists  it  will be
       overwritten at this time.

 Q  -  Terminates the utility without performing any update function.

 Press 'RETURN' to view initial selection menu

 ------------------------------------------------------------------------
```

The menus are self-explanatory. The "default values" mentioned in the help menu under item number 1 are presented when the user selects option 1 (define TT header). This illustrates an important point: It is not necessary to go through every selection item in the work menus. Most of the items will default to sensible values if they are not selected by the user. In particular, it is not necessary for the user to go though the define TT header menu. A default TT header will be defined if the user does not select this option. The user may elect to view the define TT header menu by making that selection, but it is not necessary to modify any of the default values that will be presented by that menu.

The only way to exit the TTGEN utility is to select the "Q" option when the initial selection menu is presented. Selecting the "C" option will cause a task table to be built and control to be returned to the initial selection menu. If a mistake was made it is still possible to correct it and select the "C" option again, which will overwrite the erroneous TT file with the corrected information.

2.4.3 "Define Task Table Header" Menu

The "define task table header" menu, along with its associated help menu, is shown below:

```
-------------------------------------------------------------------------

Define task table header:

 1  -  Table address (hex)..............................$not def

 2  -  Header ID........................................!HDR

 3  -  Error Handling Offset (Mnemonic or hex)...........0

 Q  -  Quit (return to initial selection menu)

 H  -  Help

Enter selection >

----------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------

                   H E L P (for Task Table Header Menu)

Selection   Description

1  -  (Optional) Enter the address that the task table will occupy in memory.
      An ORG statement will be generated reflecting this value.  If no
      selection is made, no ORG statement will be generated.

2  -  (Optional) Enter the header ID.  The ID is a 4-character alphanumeric
      string.  The default value is '!HDR'.

3  -  (Optional) Enter the error handling offset. The entry may be either hex
      (representing an absolute address where the error handling logic
      resides) or a mnemonic (representing the assembler label of the first
      instruction of the error handling code).  If no entry or an entry of 0
      is made,the task initiator will assume there is no error handling code.

Press 'RETURN' to view task table header menu

-------------------------------------------------------------------------
```

Note that all of the selections on the TT header menu are optional. This means that it is not necessary to select this menu at all from the initial selection menu. If it is not selected, all of the values defined by this menu will default.

Note the following about entering constant data: If selection 1 is chosen, TTGEN prompts for entry of the hex value representing the starting address for the TT. The value entered may have a leading "$" but it is not required. For selection 2 a prompt will ask for entry of the mnemonic representing the header ID. ASCII constants must be entered without delimiting quotes. For selection 2, if fewer than four characters are entered, the ID will be blank-filled on the right. If more than four characters are entered, the ID will be truncated from the right to four characters. For selection 3, a prompt will ask for entry of either a mnemonic (which corresponds to an assembly language label) or a hex value. In this case, the leading "$" is required for the hex input since there may be confusion in some cases. The constant 'FADE' is an assembly language mnemonic but the constant "$FADE" is a hex value.

2.4.4 "Define Single Task Entry" Menu

The "define single task entry" menu, along with its associated help menu, is shown below:

```
 ----------------------------------------------------------------------------


Define single task entry:     'ZAPP'


  1  -  Load module file name....................not def

  2  -  Task starting address (hex)......................$not def

  3  -  Task initial state (Ready 'R' or Dormant 'D').....D

  4  -  Task processing priority (decimal)...............0

  Q  -  Quit (return to initial selection menu)

  H  -  Help

                    Next available address = $not def

 Enter selection >

 ----------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------

                   H E L P (for Task Entry Menu)

Selection   Description

1  -  Enter the load module filename for the desired task.

2  -  (Optional) Enter the task start address.  This address must be in  hex
      and  represents the address that the first segment of the task will
      occupy.  If no entry is made for this selection, the starting addresses
      will be obtained from the load module file.

3  -  (Optional) Enter the task initial state. The two possible responses are
      'D' for dormant or 'R' for ready.  The default value for this selection
      is 'D'.

4  -  (Optional) Enter the task processing order.  The number entered is a
      decimal number in the range 0-32767.  The default value is 0.

Press 'RETURN' to return to view task entry menu

-----------------------------------------------------------------------
```

When selection 2 is requested from the main menu, a prompt will ask for entry of a task entry ID. The ID entered will identify the particular task entry being defined. The task entry ID for the above example is "ZAPP". While still in the utility, a particular task entry menu can always be recalled through use of the associated task entry ID. All the information previously defined for that task entry will be displayed if the menu is returned to at a later time. When the task table is built, the task entries will be inserted into the table in the order in which they were defined. This order is not necessarily the order in which TDTI will process the entries. That order is determined by the processing order (selection 4) defined for the task entry.

Note that only the load module filename is required. All of the other entries are optional. Most of the information that goes into the TT will be obtained from the LIB of the load module file. As mentioned earlier, the load module must be available for TTGEN to access. Refer to the M68000 Family Linkage Editor User's Manual for options available when linking a task.

The "next available address" line in the menu will assume a value when the task start address is defined. This value is the address of the byte of memory immediately following the last byte of the task defined. This can be used as a guide to determine where the next task may begin. TTGEN does not enforce address checking, however. It is possible to define tasks with overlapping addresses. If a task start address is defined that differs from the LIB address defined for that task, TTGEN will automatically bias the addresses of all the segments defined for that task by the appropriate amount before inserting the information into the TT. No change will be made in the load module associated with that task.

2.4.5 Modifying TTGEN Output

This utility is designed to provide a very simple interface to assist the user in coding a rather complex TT. As mentioned earlier, it may not be possible to produce the TT to support unusual options using TTGEN alone. It is possible, however, to produce a TT that is fairly close to the desired form, and use the VERSAdos editor to modify the output file that TTGEN produces. This section discusses the limitations that may be encountered in using the TTGEN utility.

The first limitation is the inability to use the utility to produce a table entry that supports special or "non-standard" processing. Each entry in the table produced by TTGEN must have an associated task. It is possible, however, to have a table entry that is not associated with any task, but merely contains a pointer to some non-standard user-written code that executes as a subroutine to the TDTI. The easiest way to handle this is to have a small dummy task defined on the system and configure the table entry using TTGEN to support that dummy task. After the table is created, the editor can be used to insert a pointer to the non-standard code into the cell that has the comment "Offset to special non-standard processing". A nonzero value here will cause TDTI to ignore all of the information relating to the dummy task (other than processing priority) and simply execute the special code as a subroutine.

There is no provision to modify the user-generated task ID. This is the ID that is inserted into events queued by the task. This field can easily be modified by editing the task entry line that has the comment "User-generated task ID".

Another limitation is the inability to specify segment directive options. TTGEN always sets the value $0100 (bit 8 set) into the segment directive options field for each segment associated with a task. The option set by TTGEN tells TDTI to allocate the segment at the defined physical address. This should satisfy most requirements. The other defined options can be invoked by simply finding the appropriate segment definition area in the output file and editing the "segment directive options" field. Refer to the M68000 Family Real-Time Multitasking Software User's Manual for further information about segment directive options.

2.5 EXAMPLE

The many menus that appear during the use of the utility preclude presenting a complete example of the use of TTGEN; therefore the following example presents only the user inputs along with some comments about the menus that are displayed. Following the dialog description, the output TT is presented.

2.5.1 Interactive Dialog

The following example represents an unrealistic system, but it illustrates the use of the utility to create a TT with two tasks defined. The first task will be TTGEN itself (a three-segment task), and the second task will be the VERSAdos assembler (a two-segment task). Both of these tasks' load modules should presently be on the VERSAdos media; this example can be performed by a user to aid in understanding the use of the utility. User inputs are shown on the left along with comments. Entries associated with lower level menus are indented.

=TTGEN Invokes the utility. Since no output file was selected, the output filename defaults to TTABLE.SA.

1 The user selects option 1 (define TT header). The TT header menu is presented.

> **1** Option 1 is selected (table address).
>
> **$2B00** The table address is defined to start at $2B00.
>
> **2** Option 2 is selected (header ID).
>
> **HEAD** The header ID is defined to be "HEAD". Note that the ID is entered without delimiting quotes.
>
> **Q** This will return control to the initial selection menu.

2 Option 2 is selected (define single TT entry).

ZAPP The utility queries the user for task entry ID. The entry ID is defined to be "ZAPP". The "define single task entry" menu is presented.

> **1** Option 1 is selected (load module filename).
>
> **0..TTGEN.LO** The filename is entered in response to the utility's query.
>
> **2** Option 2 is selected (task starting address).
>
> **3000** The starting address is defined to be $3000. Note that the leading "$" was not entered here. If it had been, the effect would have been the same.
>
> **Q** This will return control to the main menu. Note that task processing order (option 4) was not selected. It will default to a value of 0.

```
2                   Option 2 was selected to define the second task.

ZORK                The associated task entry ID is "ZORK".

    1                    Option 1 is selected (load module filename).

    0..ASM.LO            The  filename  is  entered  in  response  to  the
                         utility's query.

    2                    Option 2 is selected (task starting address).

    $1DE00               The starting address is defined to be $1DE00.

    Q                    This  returns  control  to  the initial selection
                         menu.

C                   This selection causes the task table to be built.

Q                   The utility terminates.

=
```

2.5.2 TTGEN Output

The following assembly language code was created by TTGEN from the example input just presented:

```
           PAGE
* Table driven task initiator (TDTI) startup table.
* Created with TTGEN

           ORG    $2B00

*-----------------------------------------------------------------------*
*           T A S K   T A B L E   H E A D E R                           *
*-----------------------------------------------------------------------*

           DC.L   'HEAD'          Header ID.

           DC.L   0               Address offset of error handling routine.
*                                 Zero means no error handling routine.

           DC.L   4               Address offset to 1st table entry.
*-----------------------------------------------------------------------*
*           E N D   OF   T A S K   T A B L E   H E A D E R              *
*-----------------------------------------------------------------------*
           PAGE
*-----------------------------------------------------------------------*
*                         TASK    ENTRY    ZAPP                         *
```

```
*------------------------------------------------------------------------------*
          DC.L    'ZAPP'             Task entry ID
          DC.L    ZAPP_END-*         Offset to next entry
          DC.L    $0                 Offset to special non-standard processing
          DC.W    0                  Processing order value
          DC.B    0                  Reserved
          DC.B    'D'                State ('D'=dormant, 'R'=ready)
          DC.L    'TTGE'             Taskname
          DC.L    $00000000          '....' Task session number
          DC.W    %0000000000000000  TCB directive options
          DC.L    $0                 Monitor taskname
          DC.L    $00000000          '....' Monitor session number
          DC.B    $00                Task initial priority
          DC.B    $00                Task limit priority
          DC.W    %0000100000000000  Task attributes
          DC.L    $00003000          Task start address
          DC.W    0                  User-generated task ID
          DC.W    3                  Number of segments for task

*
*------------ Segment entry for task 'ZAPP', segment 'SEG1'
*
          DC.W    %0000000100000000  Segment directive options
          DC.W    %0100000000000000  Segment attributes
          DC.L    'SEG1'             Segment name
          DC.L    $00003000          Start address of segment
          DC.L    $00005200          Length of segment

*
*------------ Segment entry for task 'ZAPP', segment 'SEG2'
*
          DC.W    %0000000100000000  Segment directive options
          DC.W    %0000000000000000  Segment attributes
          DC.L    'SEG2'             Segment name
          DC.L    $00008200          Start address of segment
          DC.L    $0000C400          Length of segment

*
*------------ Segment entry for task 'ZAPP', segment 'RRTL'
*
          DC.W    %0000000100000000  Segment directive options
          DC.W    %0101000000000000  Segment attributes
          DC.L    'RRTL'             Segment name
          DC.L    $00014600          Start address of segment
          DC.L    $00008F00          Length of segment

ZAPP_END  EQU     *

)*-----------------------------------------------------------------------------*
*                     END OF ENTRY FOR --- ZAPP                                *
```

```
*----------------------------------------------------------------------------*


         PAGE
*----------------------------------------------------------------------------*
*                        TASK    ENTRY    ZORK                               *
*----------------------------------------------------------------------------*
         DC.L   'ZORK'              Task entry ID
         DC.L   0                   Offset to next entry
         DC.L   $0                  Offset to special non-standard processing
         DC.W   0                   Processing order value
         DC.B   0                   Reserved
         DC.B   'D'                 State ('D'=dormant, 'R'=ready)
         DC.L   'RASM'              Taskname
         DC.L   $00000000           '....' Task session number
         DC.W   %0000000000000000   TCB directive options
         DC.L   $0                  Monitor taskname
         DC.L   $00000000           '....' Monitor session number
         DC.B   $00                 Task initial priority
         DC.B   $00                 Task limit priority
         DC.W   %0000100000000000   Task attributes
         DC.L   $0001DE00           Task start address
         DC.W   0                   User-generated task ID
         DC.W   2                   Number of segments for task
*
*------------ Segment entry for task 'ZORK', segment 'IASM'
*
         DC.W   %0000000100000000   Segment directive options
         DC.W   %0101000000000000   Segment attributes
         DC.L   'IASM'              Segment name
         DC.L   $0001DE00           Start address of segment
         DC.L   $00015C00           Length of segment

*
*------------ Segment entry for task 'ZORK', segment 'SEG2'
*
         DC.W   %0000000100000000   Segment directive options
         DC.W   %0000000000000000   Segment attributes
         DC.L   'SEG2'              Segment name
         DC.L   $00033A00           Start address of segment
         DC.L   $00009400           Length of segment

ZORK_END EQU    *

*----------------------------------------------------------------------------*
*                      END OF ENTRY FOR --- ZORK                             *
*----------------------------------------------------------------------------*

*----------------------------------------------------------------------------*
*                 E N D   O F   T A S K   T A B L E                          *
*                                                                            *
*                         Have a nice day !                                  *
*----------------------------------------------------------------------------*
```

THIS PAGE INTENTIONALLY LEFT BLANK.

CHAPTER 3

CREATION OF A TDTI OPERATING SYSTEM

3.1 GENERAL

There are two types of TDTI operating systems: RMS68K and VERSAdos. The VERSAdos TDTI system differs from the RMS68K version in that it includes the VERSAdos I/O subsystem and applicable drivers for the user's application. The figure below illustrates the structures of the two types of TDTI systems and the necessary user-written portion.

```
       RMS68K              VERSAdos              USER

  +---------------+   +---------------+   +--------------------+
  | RMS68K        |   | VERSAdos      |   | USER-WRITTEN       |
  +---------------+   | (IOS/FHS)     |   | INFORMATION:       |
  | TDTI          |   +---------------+   |* TASK TABLE        |
  +---------------+   | DRIVERS       |   |* NON-STANDARD CODE |
  | SYSINIT       |   +---------------+   |* ERROR HANDLING    |
  +---------------+                       |* TASK(S)           |
                                          +--------------------+
```

Because the Table Driven Operating System can operate on a target system that has a Memory Management Unit (MMU), it is necessary that the TT, any existing non-standard user code, and any user-written error handling logic reside in a single user-specified partition. TDTI will allocate a segment, using the address range for this partition, so that it can access the information contained there. The user has complete flexibility, within the address range, for his resource allocation.

In systems which are to be downloaded, a VERSAdos 4.4 limitation requires that a RAM partition be logically defined to the system as ROM to prevent downloaded information from being cleared at system boot time. The system initializer at boot time will zero out all RAM defined in the RAM partitions, but will not attempt to initialize a partition defined as ROM. Therefore, to prevent destruction of the user's application task(s), TT, and any non-standard user-specified code or error handling logic that has been downloaded, a partition defined as ROM must be in the address range of the downloaded modules. This modification must be made in the INITDAT.AG module.

The user application task(s) must also reside in a ROM partition but it does not have to be the same ROM partition that contains the TT and optional non-standard user specified code or error handling logic. No restrictions have been placed on the RMS68K or VERSAdos portions shown above, but the user-written information must be defined in one or more ROM partitions. The demonstration chainfile discussed in Chapter 4 puts these items into a single partition. Note that a partition defined as ROM can physically reside in ROM or RAM.

On VMEsystems with a MMU and cache memory (MVME120 and MVME121), partition 1 must be configured. Partition 1 contains the default memory allocated to user tasks. Allocating memory from this partition ensures that the base address will be on a required 2Kb boundary.

The above portions can be created separately but they are interrelated via address links (pointers). The following defines the relationships that must be resolved but does not address the method of implementation to satisfy those relationships.

a. TDTI has three external references defined that must be resolved. These references are defined in the user-written information and include:

XREF	TTSSA	Pointer to the TT header
XREF	AREASA	Start physical address of ROM partition
XREF	AREAEA	End physical address of ROM partition

Between the start and end addresses of this ROM partition must be at least the following:

Task table	Required
Error handling logic	Optional
Non-standard user code	Optional

Additional information may reside in this partition at the user's discretion.

b. Links from user-written code (tasks, error handling logic, and non-standard startup logic) to TDTI subroutines must be resolved if TDTI subroutines are used. This is optional, and if the user elects to write his own code instead of using TDTI subroutines, this relationship would not have to be resolved. The following external definitions have been defined in TDTI:

XDEF	TI70200	Character fill subroutine
XDEF	TI70500	Create TCB
XDEF	TI71000	Interface to error handling routine
XDEF	TI71200	Get task ID
XDEF	TI71300	Allocate task segments
XDEF	TI72800	Start specified task

Refer to Appendix D for detailed information on each subroutine.

3.2 CREATING THE SYSTEM

The steps to create an RMS68K or VERSAdos system are as follows:

a. Execute the TDTIGEN1 utility to create the arguments required as input to the chainfile 9998.TDTI.COPY.CF. The arguments, which TDTIGEN1 stores in the target *volume:user number* file RESTOREA.CF, include the target *volume:user number* where the SYSGEN will be executed, the source *volume* where the SYSGEN files reside, the VERSAdos *catalog* for the SYSGEN product, and the type of SYSGEN, RMS68K or VERSAdos. Typically the source volume and the target volume are the same, but this is not a requirement. When TDTIGEN1 is executed, an interactive dialog begins that displays current defaults and prompts the user for input of the arguments. A typical example follows:

```
=TDTIGEN1
Target Volume .......... SYS
Target User Number ..... 8007
Product Catalog    ..... VME110
System Type        ..... V
Source Volume .......... SYS

Target Volume      - Volume where SYSGEN will occur
Target User Number - User number where SYSGEN will occur
Product Catalog    - VERSAdos catalog for target SYSGEN system
Type of SYSGEN     - R = RMS or V = VERSAdos
Source Volume      - Volume where SYSGEN-related files reside

C = Configure file with current parameters and Quit
Q = Quit with no file configuration
    A carriage return retains current value

Target Volume ..........SYS >(CR)
Target User Number......8007 >9990
     .
     .
     .
```

b. From the target *volume:user number* enter the following command line:

```
=source volume:9998.TDTI.COPY.CF
```

This chainfile will copy to the target *volume:user number* those files that require modification. The chainfile will automatically modify &.CNFGTASK.CI and TDTI.CI according to the type of TDTI operating system the user has selected. This chainfile uses the SYSGEN implementation to resolve the relationship, discussed above, between the user-written information and the TDTI task. A listing of the chainfile is provided in Appendix C.

Upon completion of this chainfile the user is notified of the files that must be modified. These files are:

&.INITDAT.AG — Must be modified to reflect the user's partitioning for his SYSGENed system.

&.TDTI.CI — Must be modified to identify the start/end address of the ROM partition that contains the TT to be processed as well as the the start address of the TT in the partition (an example is provided in Appendix B).

catalog.CNFGDRVR.CI — If this is a TDTI RMS68K operating system, all references to any drivers must be removed. If this is a TDTI VERSAdos operating system, the user should reference only those drivers required for his application (an example is provided in Appendix B).

c. Initiate the command line:

```
=TDTI.PRODUCT.CF
```

from the target *volume:user number*. The results of this chainfile will be the type of TDTI system requested by the user. A listing of this chainfile is provided in Appendix C.

NOTE

Due to a VERSAdos 4.4 limitation, IOS, FHS, and GET.TASKID.AG contain an error which prevents the system from booting if a RAM system was SYSGENed without the File Management System (FMS). The corrected modules, which the user should apply to his system, reside on the latest quarterly update media.

CHAPTER 4

TDTI/VERSAdos SYSTEM EXAMPLE PROGRAM

4.1 EXAMPLE CHAINFILE

The VERSAdos software includes, under user number 9990, a chainfile which will create an example of a Table Driven Task Initiator (TDTI)/VERSAdos operating system containing four tasks, each outputting a message to device CN00. This chainfile, TDTIVDOS.EXAMPLE.CF, will perform all of the steps necessary to create a TT, application tasks to use with the system, non-standard user code, and error handling logic. The example chainfile target system is an MVME110. The demonstration chainfile identifies the steps required to create a table driven operating system. The implementation scheme, addresses chosen, and target system used in this example are not binding on the user but merely illustrate one way to create a TDTI/VERSAdos system.

Listings of files used in this example are provided in appendices, as follows:

Appendix A -- Example Task Table
Appendix B -- Files Modified for Example
Appendix C -- Example Chainfiles

The creation of the TDTI/VERSAdos system by the example chainfile TDTIVDOS.EXAMPLE.CF has two major steps:

1. Creation of the TDTI system.
2. Creation of the user-responsible code.

Step 1 involves the following:

a. Executing the TDTIGEN1 utility in the target *volume:user number* where the SYSGEN will occur, to establish the arguments for the execution of the chainfile TDTI.COPY.CF.

b. Initiating the chainfile *volume*.9998.TDTI.COPY.CF to copy and identify for the user those files that require user modification before the TDTI system can be built. The chainfile TDTIUSER.MODIFY.CF reflects these changes.

c. Making modifications to the files that define partitioning, identify where the TT resides, and identify the driver that is applicable to this application. Files that are modified include:

&.INITDAT.AG
&.TDTI.CI
catalog.CNFGDRVR.CI

d. Initiating the chainfile TDTI.PRODUCT.CF. Upon completion, this chainfile will produce the S-record file VME110.VERSADOS.MX, which can be downloaded.

Step 2 involves the following:

a. Creating the user-required application tasks which are to be executed. The chainfile TDTI.APLICATN.CF reflects this information. The files related to this application are TSK1.SA, TSK2.SA, TSK3.SA, and TSK4.SA.

b. Creating the TT and any optional error handling logic or non-standard code defined by the user. The error handling logic is in module TTSEHL.SA, the TT is in TTS.SA, and the non-standard user-written code is in modules TE01USC.SA and TE05USC.SA. The implementation to resolve the relationships between user-written code and the TDTI referenced routines was a SYSGEN. The SYSGEN modules involved are TTSUCEHL.SYSGEN.CF and TTSUCEHL.CD. All of these modules reside in user number 9990.

The chainfile TDTITTS.EXAMPLE.CF reflects this information and the SYSGEN technique used to collect this information into a single module.

4.2 EXECUTION OF EXAMPLE CHAINFILE

To execute the example chainfile, perform the following steps:

a. Log on to the *volume:user number* where the SYSGEN is to be executed.

b. Execute the TDTIGEN1 utility to define the arguments for the chainfile. The VERSAdos catalog for the target system must equal "VME110" and the type of TDTI system to be created must equal "V", which is the VERSAdos system. The other arguments requested by the utility are user-selectable.

c. Initiate the chainfile 9990.TDTIVDOS.EXAMPLE.CF. This chainfile will make calls to the following chainfiles from the target *volume:user number:*

```
9998.TDTI.COPY.CF
     TDTIUSER.MODIFY.CF
     TDTI.PRODUCT.CF
     TDTI.APLICATN.CF
     TDTITTS.EXAMPLE.CF
```

After the chainfiles above have been executed, the user can download all of the files with an extension of ".MX" to the MVME110 system, set the PC and the stack register A7, and initiate the process with the GO command. Refer to Appendix B for information regarding the startup value.

4.3 CHAINFILE DESCRIPTIONS

The following paragraphs explain the function of the chainfiles called by the chainfile 9990.TDTIVDOS.EXAMPLE.CF (refer to paragraph 4.2). Refer also to Appendix C, which contains listings of these chainfiles.

4.3.1 TDTI.COPY.CF

This chainfile will copy the files that need to be modified to the target *volume:user number*. For demonstration purposes, pre-modified files exist, and are copied to the target *volume:user number*. These pre-modified files reflect the configuration of the target MVME110 system. The files copied are:

9990.EXAMPLE.INITDAT.AG	Defines the example partitioning
9990.EXAMPLE.TDTI.CI	Defines partitioning addresses
9990.EXAMPLE.CNFGDRVR.CI	Contains only the ACIA driver

4.3.2 TDTIUSER.MODIFY.CF

The changes the user would make are reflected in this chainfile. For this example, the following conditions exist:

RAM partition $40000 - $9FFFF
ROM partition $300000 - $31EFFF
ROM partition $31F000 - $31FFFF
Task table start address - $300A00
Only the ACIA driver exists in the system

4.3.3 TDTI.PRODUCT.CF

This chainfile will perform the following:

a. Save the files modified by the user so they will not be destroyed.

b. Initiate an MVME110 COPYSGEN process to copy all files required for the SYSGEN to the target *volume:user number*.

c. Re-establish the modified files as the ones to use during the SYSGEN.

d. Copy the TDTI-related files to the target *volume:user number*. The files copied are:

9998.&.TDTI.LG
9998.&.TDTI.RO
9998.&.TDTIRSL.CI
9998.&.TDTIVU.CI
9998.TTSSA.TDTI.AG

e. Edit the VERSADOS.CD module to include the TDTI command modules TDTI.CI and TDTIRSL.CI. The TDTI.CI will define the task attributes for the TDTI load module created at SYSGEN time. The TDTIRSL.CI module and the function it serves are explained under the discussion of the TDTITTS.EXAMPLE.CF chainfile (refer to paragraph 4.3.3).

f. Initiate the SYSGEN process to produce the TDTI/VERSAdos system for the MVME110 and build the corresponding module to be downloaded. A by-product of this SYSGEN is the file XTDTIVU.CI, whose significance will be explained in the discussion of the TDTITTS.EXAMPLE.CF chainfile (refer to paragraph 4.3.5).

4.3.4 TDTI.APLICATN.CF

This chainfile reflects the application the user has defined for his system. As stated previously, there are four tasks outputting a message to device CN00 in a never-ending loop. This chainfile assembles, links, and creates the modules to be downloaded to the target system.

4.3.5 TDTITTS.EXAMPLE.CF

This chainfile creates the module containing error handling logic, non-standard user-specified code, and the TT. The TT is required, while non-standard user-specified code and the error handling logic are optional.

This module could have been created in any number of ways, but for this demonstration a SYSGEN process was chosen. This implementation was chosen to resolve the address references to the TDTI subroutines used by the non-standard code in task entry 2.

For this example chainfile, TT entries 2 and 5 were selected to have non-standard user-specified code. Task entry 2 accesses routines used by TDTI to perform "standard" processing, but the additional function of starting the task with specific address and data register values was added. Since task entry two uses TDTI routines, it is necessary to know where the TDTI task started and to include the TDTI.RO module at link time to satisfy external references to those subroutines referenced by task entry 2. The command include file TDTIRSL.CI, which was executed at SYSGEN time for the TDTI operating system, created the module XTDTIVU.CI. This module identifies the start address of the TDTI module. (Refer to Appendix B.)

Task entry 5 performs none of the "standard" processing functions, but instead initiates the RMS68K directive to delay for five seconds before continuing to process the balance of the TT.

The chainfile copies the following to the target *volume:user number*:

a. All non-standard user-specified code

b. The user-specified error handling logic

c. Task table-related files

d. The files needed to execute the SYSGEN for this module (refer to Appendix B).

After the files are copied, assemblies are performed on the error handling logic, the non-standard user-specified code, and the TT. The SYSGEN is initiated, and upon its completion the corresponding file to be downloaded is created.

THIS PAGE INTENTIONALLY LEFT BLANK.

APPENDIX A

EXAMPLE TASK TABLE

A.1 EXAMPLE OF TASK TABLE HEADER

```
TTSSA:
         DC.L      '!TTS'
*                            User-generated task entry identifier.  Eye
*                                 catcher only -- No function associated with
*                                 this field.
         DC.L      TTSEHL-*
*                            Self-relative offset to the error handling
*                                 logic to be executed if the TDTI task
*                                 encounters an error while processing the
*                                 TTS.
*                            If the user elects to return from the error
*                                 handling logic, via an RTS instruction, he
*                                 must preserve the stack pointer.
                                  Preservation of all other registers will be
*                                 maintained by the Table Driven Task
*                                 Initiator task.  Upon return, processing
*                                 will commence with the next TTE.
*                            If this field is zero the error handling logic
*                                 will not be executed. If an error occurs,
*                                 current TTE will be skipped and processing
*                                 will resume with the next TTE.
*                            If error handling logic exists and an error is
*                                 encountered, but bit zero of this field is
*                                 set, processing on the current TTE will be
*                                 terminated and the error handling logic will
*                                 not be executed. Processing will commence
*                                 with the next TTE.

         DC.L      TEID0001-*
*                            Self-relative offset to the first TTS entry.
*                                 A value of zero implies that there are no
*                                 TTEs.
```

A.2 EXAMPLE OF "STANDARD" TASK TABLE ENTRY

```
TEID0002:
         DC.L      'TE02'
*                            User-generated task entry identifier.  Eye
*                                 catcher only -- no function associated with
*                                 this field.
         DC.L      TEID0003-*
*                            Self-relative offset to the next TTE in the TTS.
*                                 A value of zero implies there are no more
*                                 TTEs.

         DC.L      0
*                            Self-relative offset to the non-standard
*                                 code to be processed for this task.
*                            A value of zero or an odd value (bit zero set)
*                                 means no non-standard code will be
*                                 executed.
*                            If non-standard code is to be executed a
*                                 Jump-to-Subroutine (JSR) call will be
*                                 made to the user-written non-standard code.
*                                 The user is responsible for maintaining the
*                                 stack pointer and returning via an RTS
*                                 instruction. Register preservation for
*                                 registers other than the stack pointer will
*                                 be done by the TDTI task. On a call to the
*                                 non-standard code register, A5 will point to
*                                 the TT header, and register A4 will point to
*                                 the TTE to be processed.
*                                 The user is responsible for performing all
*                                 actions normally done by the TDTI task.
*                                 Routines normally accessed by the TDTI task
*                                 will be accessible to the non-standard code
*                                 Upon return from the non-standard code the
*                                 TDTI task will commence processing with the
*                                 next task.
*                            Since the offset will always be an even value,
*                                 bit zero was chosen so the user could easily
*                                 alter the flag and not destroy the offset
*                                 value.
*                            This implementation will allow the user to easily
*                                 execute or not execute his non-standard
*                                 code.
         DC.W      5
*                            This word contains the task processing order.
*                                 The execution order is from low to high.
*                                 If a TTS had four TTEs as follows:
*
*                                        TTS          PROCESSING ORDER
*                                       ENTRY              VALUE
*                                         1                 31
*                                         2                 10
*                                         3                 17
*                                         4                 35
```

```
*
*                                The TTE processing would be TTE2, TTE3,
*                                TTE1, followed by TTE4.
*                                If multiple Task Table Entries with the same
*                                processing order value exist, the Task Table
*                                Entries will be processed sequentially from
*                                the first entry to the last entry.
*                                If a TTS had four TTEs as follows:
*
*                                       TTS          PROCESSING ORDER
*                                      ENTRY               VALUE
*                                        1                  31
*                                        2                  05
*                                        3                  17
*                                        4                  05
*
*                                the TTE processing would be TTE2, TTE4,
*                                TTE3, followed by TTE1.
*                           If this field has bit 15 set, this task
*                                entry will be ignored and the next task
*                                table entry will be processed.
*                           Since the processing order will never be a
*                                negative number, bit 15 was chosen so
*                                the user could easily alter the flag and
*                                not destroy the processing order value.
        DC.B      0
        DC.B      'R'
*                           This word contains the code of the initial
*                                state that this task is to have.
*                           Valid codes are:
*                                'R'  Ready to execute
*                                'D'  This task is dormant
*                           If the initial state does not have a valid value
*                                it will default to dormant, 'D'.
*                           The high order byte of this word has been
*                                reserved for future use.
*
*                                     +----------+----------+
*                                     | Future   | State    |
*                                     | Use      | Code     |
*                                     +----------+----------+
*
        DC.L      'TSK2'
*                           Task name
        DC.L      '0020'
*                           Task session number
        DC.W      0
*                           TCB directive options
        DC.L      0
*                           Monitor task name
        DC.L      0
*                           Monitor task session number
```

```
        DC.B    $42
*                                   Task initial priority
        DC.B    $7F
*                                   Task limit priority
        DC.W    0
*                                   Task attributes
        DC.L    $312000
*                                   Task entry point address
        DC.W    0
*                                   User generated identification for task
        DC.W    2
*                                   Number of segments associated with task

        DC.W    $0100
*                                   Segment Directive Options
*                                   Bit     Meaning
*                                    13     0 - An address is specified in the
*                                               address field for this segment.
*
*                                           1 - RMS68K supplies logical address
*                                               equal to physical address.
*
*                                    12     0 - Shareable segment is not permanent.
*
*                                           1 - Shareable segment is permanent.
*
*                                     8     0 - RMS68K does not attempt to allocate
*                                               the segment at the physical address
*                                               specified in the address field for
*                                               this segment.

*                                           1 - RMS68K attempts to allocate the
*                                               segment at the physical address
*                                               specified in the address field
*                                               for this segment.
*                                               Since this is a physical address,
*                                               by definition logical address will
*                                               equal physical address.

        DC.W    0
*                                   Segment Attributes
*                                   Bit     Meaning
*                                    14     0 - Segment is to be read/write.

*                                           1 - Segment is to be read only.
*
*                                    13     0 - Segment is not locally shareable.

*                                           1 - Segment is locally shareable.
*
*                                    12     0 - Segment is not globally shareable.

*                                           1 - Segment is globally shareable.
```

```
         DC.L    'SEG0'
*                          Segment name
         DC.L    $312000
*                          Segment address
         DC.L    $200
*                          Segment size in bytes
         DC.W    $1100
*                          Segment Directive Options
*                          Bit      Meaning
*                           13      0 - An address is specified in the
*                                       address field for this segment.
*
*                                   1 - RMS68K supplies logical address
*                                       to equal physical address.
*
*                           12      0 - Shareable segment is not permanent.
*
*                                   1 - Shareable segment is permanent.
*
*                            8      0 - RMS68K does not attempt to allocate
*                                       the segment at the physical address
*                                       specified in the address field for
*                                       this segment.

*                                   1 - RMS68K attempts to allocate the
*                                       segment at the physical address
*                                       specified in the address field
*                                       for this segment.
*                                       Since this is a physical address,
*                                       by definition logical address will
*                                       equal physical address.

         DC.W    $1000
*                          Segment Attributes
*                          Bit      Meaning
*                           14      0 - Segment is to be read/write.

*                                   1 - Segment is to be read only.
*
*                           13      0 - Segment is not locally shareable.

*                                   1 - Segment is locally shareable.
*
*                           12      0 - Segment is not globally shareable.

*                                   1 - Segment is globally shareable.
*
```

```
*                             11       0 - Segment is not memory mapped I/O
*                                          space.
*
*                                      1 - Segment is memory mapped I/O
*                                          space. The address given in the
*                                          address field for this segment
*                                          must be a physical address that is
*                                          not in the limits of allocatable
*                                          RAM. If this bit is set, none of
*                                          the GTSEG options are applicable
*                                          and the segment is allocated as
*                                          a shared segment.
         DC.L      'SEGC'
*                            Segment name
         DC.L      $31F000
*                            Segment address
         DC.L      $100
*                            Segment size in bytes
```

APPENDIX B

FILES MODIFIED FOR EXAMPLE

B.1 TDTI.CI MODULE

```
 MSG      *************************************************************
 MSG      *                                                           *
 MSG      *  Link the TDTI task                                       *
 MSG      *                                                           *
 MSG      *************************************************************

 AREASA   =         $????????          Partition start address of segment
*                                      containing
*                                           Task Table Structure     Required
*                                           Error Handling Logic     Optional
*                                           Non-Standard Code        Optional
 AREAEA   =         $????????          Partition end address for above segment
 TTSSA    =         $????????          Task Table start address


 STDTISA  = \PC

 TASK       &.TDTI.LO
 STATE    = 'READ'
 ATTRIB   = 'RTIM'
 ATTRIB   = 'CRIT'
 ATTRIB   = 'SYST'
 SESSION  = 1
 PRIORITY = $C8
 SUBS       TTSSA.TDTI.AG
 ASM        TTSSA.TDTI.AG,TTSSA.TDTI.RO,\ASMLS;R
 IFEQ        \ASMLSW
   =COPY       \ASMLS,\WORKLS;A
 ENDC
 SUBS       &.TDTI.LG
 LINK       &.TDTI.LG
 IFEQ     \LINKLSW
          =COPY     \LINKLS,\WORKLS;A
 ENDC
 END      TDTI
 *
 *
 *
```

Modifications to TDTI.CI Module

```
=/*
=/*   Edit the TDTI.CI module to establish the following:
=/*
=/*      Address range of the ROM partition defined for this example
=/*           This ROM partition must contain at a minimum
=/*                The Task Table                     Required
=/*                Non-standard user-specified code Optional
=/*                User-written error handling code Optional
=/*      Start address, in this address range, of the Task Table
=/*
=E &.TDTI.CI
F /AREASA/
C /$????????/$00300000/
F /AREAEA/
C /$????????/$0031EFFF/
F /TTSSA/
C /$????????/$00300A00/
QUIT
```

B.2 VME110.CNFGDRV.CI MODULE

```
=/*
=/*      Edit VME110.CNFGDRVR.CI
=/*
=/*      Remove drivers not required for this application.
=/*
=E VME110.CNFGDRVR.CI
F /NORWIN    = 1/
C /NORWIN    = 1/NORWIN    = 0/
F /NVME315   = 1/
C /NVME315   = 1/NVME315   = 0/
F /NVME320   = 1/
C /NVME320   = 1/NVME320   = 0/
F /NVME4205  = 1/
C /NVME4205  = 1/NVME4205  = 0/
F /NVME400   = 1/
C /NVME400   = 1/NVME400   = 0/
F /NVME410   = 1/
C /NVME410   = 1/NVME410   = 0/
QUIT
```

B.3 EXCERPT FROM VME110.SYSLIST.LS IDENTIFYING STARTING ADDRESS, $49700, OF TDTITASK AND STARTUP ADDRESS, $4AB00, FOR THE SYSTEM

```
FILE NAME     TASK  PROC  SEG   ADDR       TCB

RMS.LO              RMS   RMS0  $040000
                          RMS2  $040100
DRVLIB.LO           DRVL  DRVL  $044E00
TERMLIB.LO          TERM  TERM  $045000
ACIADRV.LO          ACIA  ACIA  $046300
FHS.LO        .FHS        .FHS  $046600      $ 047A00
IOS.LO        .IOS        .IOS  $047C00      $ 049500
TDTI.LO       TDTI        TDTI  $049700      $ 049C00
IOI.LO        .IOI        IOSG  $049E00      $ 04A900
                          .IOI  $04A500
SYSINIT.LO          SYSI  .INT  $04AB00

- FINAL PC VALUE = $04B500
- START-UP ADDRESS = $04AB00
```

B.4 TTSUCEHL.SYSGEN.CF RELATED MODULES

```
=/**********************
=/* TTSUCEHL.SYSGEN.CF *
=/**********************
=OPT K,N
=/****************************************************************************
=/*
=/*      SYSGEN for
=/*                Task Table                    Required
=/*                Non-Standard User Code        Optional
=/*                Error Handling Logic          Optional
=/*
=OPT J,-N
=TIME
=SYSGEN &.TTSUCEHL.CD,\1:\2/&.TTSUCEHL.SY,&.TTSUCEHL.LS
=/*
=OPT -N
=/****************************************************************************
=/**
=/**  Sysgen Completed --
=/**
=/****************************************************************************
=/****************************************************************************
=OPT -K
=END
```

B.4.1 TTSUCEHL.CD -- SYSGEN Command File for Sample System

```
*
*        TTSUCEHL.CD
*
*        This command file is used to accumulate the following modules
*
*                  Task Table                    Required
*                  Non-standard user code        Optional
*                  Error Handling Logic          Optional
*
*        The addresses should reflect those to be processed by the
*        Table Driven Task Initiator task (TDTI).
*

INCLUDE &.XTDTIVU.CI
*
*        Start address of TDTI task
*
TDTISTRT =         \TDTISA

*        The following flags are defined as follows
*
```

```
*        Non-zero  --> The information          exists
*        Zero      --> The information does not exist
*
FLAGEHL  =         1         Flag for Error Handling Logic
FLAGUSC  =         1         Flag for User-Specified Code


IFNE     \FLAGEHL

MSG      ****************************************************
MSG      *                                                  *
MSG      * Process the Error Handling Logic                 *
MSG      *                                                  *
MSG      ****************************************************

INCLUDE &.TTSEHL.CI

ENDC


IFNE     \FLAGUSC

MSG      ****************************************************
MSG      *                                                  *
MSG      * Process User-Specified Code                      *
MSG      *                                                  *
MSG      ****************************************************

INCLUDE &.TE01USC.CI
INCLUDE &.TE05USC.CI

ENDC


MSG      ****************************************************
MSG      *                                                  *
MSG      * Process the Task Table Structure                 *
MSG      *                                                  *
MSG      ****************************************************

INCLUDE &.TTS.CI


         END       SYSGEN
```

B.4.2 TTSEHL.CI -- SYSGEN Include File for Error Handling Code

```
*
*        TTSEHL.CI
*
*        This command file has been built to create a task table
*        Error Handling Logic module.
*
*        STARTEHL = Start address of the Error Handling Logic. This address
*                       is in the range of AREASA to AREAEA defined in module
*                       TDTI.CI
*

PC        =          $300000


STARTEHL = *

 SUBS        &.TTSEHL.LG
 LINK        &.TTSEHL.LG
 PROCESS     &.TTSEHL.LO
 END         TTSEHL
 *
 *
 *
```

B.4.3 TTSEHL.LG -- SYSGEN Link File for Error Handling Code

```
=/*
=/*      TTSEHL.LG
=/*
=/*      Link chainfile to create TTSEHL.LO
=/*
=/*
=LINK ,TTSEHL.LO,TTSEHL.LL;HAMIX
SEGMENT EHLO:1     \PC
INPUT &.TTSEHL.RO
END
=/*
=END
```

B.4.4 &.TE01USC.CI -- SYSGEN Include File for Task 1

```
*
*        &.TE01USC.CI
*
*        This command file has been built to create a non-standard code
*        module for use with the Table Driven Task Initiator task.
*
*        SUSCTE01 = Start address of the User-Specified Code. This address
*                   is in the range of AREASA to AREAEA defined in module
*                   TDTI.CI
*
*


PC = $300500

SUSCTE01 = *

 EXCLUDE    TDTI
 SUBS       &.TE01USC.LG
 LINK       &.TE01USC.LG
 PROCESS    &.TE01USC.LO
 END        TE01USC
 *
 *
 *
```

B.4.5 TE01USC.LG -- SYSGEN Link File for Task 1

```
=/*        TE01USC.LG
=/*
=/*        Link file to link non-standard code for this task entry
=/*
=LINK ,&.TE01USC.LO,&.TE01USC.LL;HAMIX
 SEG TDTI:0 \TDTISTRT
 SEG T01U:2 \PC
 IN  TTSSA.TDTI.RO
 IN      &.TDTI.RO
 IN      &.TE01USC.RO
 END
=END
```

B.4.6 &.TTS.CI -- SYSGEN Include File for Task Table

```
*
*        &.TTS.CI
*
*        This command file has been built to create the task table
*        module for use with the Table Driven Task Initiator task.
*
*        STARTTTS = Start address of the Task Table Structure. This address
*                        is in the range of AREASA to AREAEA defined in module
*                        TDTI.CI
*

PC = $300A00

 STARTTTS = *

 SUBS       &.TTS.LG
 LINK       &.TTS.LG
 PROCESS    &.TTS.LO
 END
 *
 *
 *
```

B.4.7 TTS.LG -- SYSGEN Link File for Task Table

```
=/*
=/*      TTS.LG
=/*
=/* Link chainfile to create TTS.LO
=/*
=LINK ,&.TTS.LO,TTS.LL;AHMIX
DEFINE   TTSEHL,\STARTEHL
DEFINE   TE01USC,\SUSCTE01
DEFINE   TE05USC,\SUSCTE05
SEG TTS0:4 \PC
INPUT     &.TTS.RO
END
=/*
=END
```

B.4.8 TE05USC.SA -- Sample User Code That Does Not Initiate Task

```
         PAGE

         NOLIST
         INCLUDE          &.TR1.EQ
         LIST

         PAGE

*
*        External definitions
*

         XDEF      TE05USC
         PAGE
*
*        It should be noted that this user-specified code has been used to
*        execute a delay for 5 seconds before returning control back
*        to the TDTI task. This example of user-specified code has been
*        included to illustrate that there are other functions that the
*        user specified code could be used for other than performing the
*        standard TDTI functionality. This particular usage provides the
*        the user the capability to complement the order in which tasks are
*        executed by adjusting the timing of their execution.
*
         PAGE

         SECTION   3

*
*        TE05 User Specified Code
*
*        INPUT
*
*                  A4 = Address of TTE to be processed
*                  A5 = Address of TTS
*


TE05USC:
         MOVE.L    #(5*1000),A0
*                            A0= Number of milliseconds to delay
         MOVE.L    #DELAY,D0
*                            D0 = Directive for this request
         TRAP      #1
*                            Initiate request
*                                      DELAY
         RTS
*                            Return to TDTI task
         END
```

B.4.9 TDTIRSL.CI -- SYSGEN Include File Used for Sample System

```
 SUBS    &.TDTIVU.CI
*                             Substitute into the TDTI VersaUser command file
*                             which will be used later on in the example by
*                             the SYSGEN that creates the VERSAdos module and
*                             the SYSGEN that creates the Task Table Structure
*
*                             The results of the substitution is the module
*                             XTDTIVU.CI
*
```

B.4.10 TDTIVU.CI

```
*
*       The values defined in this file will be used by the
*       VERSAdos SYSGEN and the
*       Task Table SYSGEN
*

TDTISA   =       \STDTISA
*                Starting address where the Table Driven Task Initiator
*                task is located
```

APPENDIX C

CHAINFILES USED IN EXAMPLE

C.1 TDTI.COPY.CF

```
=OPTION K
=/*
=/*     OPTION K means:
=/*              Do not translate LOWER case to UPPER case

=/*
=/*     TDTI.COPY.CF
=/*
=/*     Chainfile to perform the preliminary steps required prior to
=/*     executing the chainfile TDTI.PRODUCT.CF which will generate an 'RMS'
=/*     or a 'VERSAdos' Table Driven Task Initiator (TDTI) system.
=/*
=/*     Completion of this chainfile will identify the files to be modified
=/*     by the user prior to user invocation of the TDTI.PRODUCT.CF
=/*     chainfile.
=/*

=/*     CHAIN INVOCATION
=/*
=/*     Log on to volume:user number where SYSGEN will be executed
=/*     Respond to prompt with <Source Volume>:9998.TDTI.COPY.CF
=/*
=/*     Press RETURN to continue
=/&


=/*     Chainfile Assumptions
=/*
=/*     1. The user has executed the TDTIGEN1 utility, which creates the
=/*        file RESTOREA.CF which is REQUIRED input for this chainfile.
=/*        The file RESTOREA.CF contains the following:
=/*              Name of Source Volume which contains all the files
=/*               required to execute this chainfile
=/*              Target volume:user number where SYSGEN will occur
=/*              VERSAdos target system catalog identification for the
=/*               target system (VM04, VME110, etc.)
=/*              Type of sysgen: RMS or VERSAdos
=/*
=/*     2. The file RESTOREA.CF resides in the volume:user number where
=/*        this chainfile will execute.
=/*
=/*     Press RETURN to continue
=/&
```

```
=/*
=/*      Establish the original arguments for this chainfile
=/*
=/@ &.RESTOREA.CF

=/*
=/*      Chainfile assumptions based on file RESTOREA.CF
=/*
=/*      1. Chainfile will use \1:\2.\3
=/*         to execute the SYSGEN
=/*      2. All files required to execute this chainfile reside on
=/*         volume \5 and will be copied to \1:\2
=/*      3. The type of sysgen is for
=/IFEQ "R"\4
=/*         ’RMS’.
=/ENDIF
=/IFEQ "V"\4
=/*         ’VERSAdos’.
=/ENDIF
=/*
=/&      Press RETURN to continue      Press BREAK to terminate

=/*
=/*      Establish volume:user number as \1:\2
=/*
=USE \1:\2.&

=/*
=/*      Copy chainfiles to be modified to \1:\2.&
=/*
=COPY \5:9998.&.CNFGTASK.CI            \1:\2;YC
=COPY \5:9998.&.INITDAT.AG             \1:\2;YC
=COPY \5:9998.&.TDTI.CI                \1:\2;YC
=COPY \5:9998.\3.CNFGDRVR.CI           \1:\2;YC
=COPY \5:9998.TDTI.PRODUCT.CF          \1:\2;YC

=/IFEQ "R"\4
=/*
=/*      Build &.CNFGTASK.CI for the RMS sysgen
=/*
=/*      Remove the following from the system
=/*           I/O Request
=/*           File Handling
=/*           File Management
=/*           Session Control
=/*           Loader
=/*      For the RMS SYSGEN these modules are not needed.
=/*
=E &.CNFGTASK.CI
F /FHS$IOS$  = 1/
C /FHS$IOS$  = 1/FHS$IOS$  = 0/
F /FMS$      = 1/
C /FMS$      = 1/FMS$      = 0/
F /EET$      = 1/
```

```
C /EET$      = 1/EET$      = 0/
F /LDR$      = 1/
C /LDR$      = 1/LDR$      = 0/
QUIT
=/ENDIF

=/IFEQ "V"\4
=/*
=/*      Build &.CNFGTASK.CI for the VERSAdos sysgen
=/*
=/*      Remove the following from the system
=/*           File Management
=/*           Session Control
=/*           Loader
=/*      For the VERSAdos SYSGEN these modules are not needed.
=/*
=E &.CNFGTASK.CI
F /FMS$      = 1/
C /FMS$      = 1/FMS$      = 0/
F /EET$      = 1/
C /EET$      = 1/EET$      = 0/
F /LDR$      = 1/
C /LDR$      = 1/LDR$      = 0/
QUIT

=/*
=/*      Place the TDTI task in the dormant state. The IOI task will
=/*      START this task prior to terminating itself.
=/*
=E &.TDTI.CI
F /READ/
C /READ/DORM/
QUIT

=/ENDIF

=/*
=/*      The following files require modification prior to invoking
=/*      \1:\2.TDTI.PRODUCT.CF
=/*
=/*      \1:\2.&.INITDAT.AG
=/*      \1:\2.&.TDTI.CI
=/*      \1:\2.\3.CNFGDRVR.CI
=/*
=/*      \1:\2.&.INITDAT.AG
=/*                             Must be modified to reflect the user's
=/*                             partitioning
=/*      \1:\2.&.TDTI.CI
=/*                             Must be modified to identify the start/end
=/*                             address of the ROM partition that contains
=/*                             the task table to be processed as well as the
=/*                             start address of the task table in the
=/*                             partition
=/*
```

```
=/IFEQ "R"\4
=/*     \1:\2.\3.CNFGDRVR.CI
=/*                         All references to any drivers must be removed
=/ENDIF
=/IFEQ "V"\4
=/*     \1:\2.\3.CNFGDRVR.CI
=/*                         The user should reference only those drivers
=/*                         required for his application.
=/ENDIF
=/&     Press RETURN to continue

=/*
=/*     When the user has modified the above files he may initiate the
=/IFEQ "R"\4
=/*     RMS sysgen by invoking the following command line
=/ENDIF
=/IFEQ "V"\4
=/*     VERSAdos sysgen by invoking the following command line
=/ENDIF
=/*     \1:\2.TDTI.PRODUCT.CF

=END
```

C.2 DTIUSER.MODIFY.CF

```
=/*
=/*     TDTIUSER.MODIFY.CF
=/*
=/*     This chainfile reflects changes the user would have to make
=/*     to satisfy the needs required for his application. In our example
=/*     the files have already been modified and exist in user number
=/*     9990. This chainfile merely copies the already modified modules
=/*     to the target sysgen user number.
=/*
=/&     PRESS RETURN to continue

=/*
=/*     Restore the arguments established by TDTI.COPY.CF
=/*
=/@ &.RESTOREA.CF

=COPY \5:9990.EXAMPLE.INITDAT.AG,\1:\2.&.INITDAT.AG;YC
=COPY \5:9990.EXAMPLE.TDTI.CI,\1:\2.&.TDTI.CI;YC
=COPY \5:9990.EXAMPLE.CNFGDRVR.CI,\1:\2.VME110.CNFGDRVR.CI;YC

=END
```

C.3 TDTI.PRODUCT.CF

```
=/*
=/*      TDTI.PRODUCT.CF
=/*

=/*
=/*      Establish the original chainfile arguments established when
=/*      TDTI.COPY.CF was executed
=/*
=/@ &.RESTOREA.CF

=/*      Chainfile to produce
=/IFEQ "R"\4
=/*      an RMS
=/ENDIF
=/IFEQ "V"\4
=/*      a VERSAdos
=/ENDIF
=/*      Table Driven Task Initiator (TDTI) system.
=/*

=/*      CHAIN INVOCATION
=/*
=/*      Log on to volume:user number where SYSGEN will be executed.
=/*      Respond to prompt with TDTI.PRODUCT.CF
=/*

=/*
=/*      Chainfile ASSUMPTIONS:
=/*
=/*      1. Chainfile will use \1:\2.\3
=/*         to execute the SYSGEN
=/*      2. All files required to execute this chainfile reside on
=/*         volume \5 and will be copied to \1:\2
=/*      3. The type of sysgen is for
=/IFEQ "R"\4
=/*         'RMS'.
=/ENDIF
=/IFEQ "V"\4
=/*         'VERSAdos'.
=/ENDIF
=/*      4. File TDTI.COPY.CF has already been executed.
=/*
=/&      Press RETURN to continue       Press BREAK to terminate

=/*
=/*      Establish the target volume:user number.catalog  \1:\2.\3
=/*
=USE \1:\2.\3

=/*
=/*      Save the files that have been modified so that they are not destroyed
```

```
=/*
=COPY &.INITDAT.AG,&.INITDAT.GA;Y
=COPY \3.CNFGDRVR.CI,\3.CNFGDRVR.IC;Y

=/*
=/*      Perform the COPYSGEN for the target system
=/*
=/@ \5:9998.\3.COPYSGEN.CF \5,\1,\2

=/*
=/*      Establish the original chainfile arguments established when
=/*      TDTI.COPY.CF was executed
=/*
=/@ &.RESTOREA.CF

=/*
=/*      Re-establish the files modified by the user when TDTI.COPY.CF
=/*      was executed
=/*
=COPY &.INITDAT.GA,&.INITDAT.AG;YC
=COPY \3.CNFGDRVR.IC,\3.CNFGDRVR.CI;YC

=/*
=/*      Copy the TDTI related modules to the target volume:user number
=/*      \1:\2
=/*
=COPY \5:9998.&.TDTI.LG             \1:\2;YC
=COPY \5:9998.&.TDTI.RO             \1:\2;YC
=COPY \5:9998.&.TDTIRSL.CI          \1:\2;YC
=COPY \5:9998.&.TDTIVU.CI           \1:\2;YC
=COPY \5:9998.TTSSA.TDTI.AG         \1:\2;YC

=/*
=/*      Edit \1:\2.&.VERSADOS.CD for the SYSGEN
=/*
=/*      Merge the command file for the task initiator module and the
=/*      command module TDTIRSL.CI whose output, &.XTDTIVU.CI, will be used
=/*      in the SYSGEN for the Task Table.
=/*
=E &.VERSADOS.CD
F /FHS$IOS$/
UP 1
MERGE &.TDTI.CI
F /= \&SERFLAG/
D 1
MERGE &.TDTIRSL.CI
QUIT

=/IFEQ "V"\4
=/*
```

```
=/*      The VERSAdos 4.4 version of IOS and FHS and GET.TASKID.AG modules had
=/*      an error that occurred at boot time if the SYSGEN was for a RAM
=/*      system without the file handling module (FMS). These modules have
=/*      been corrected to eliminate that problem and must overlay the modules
=/*      copied over during the copysgen.
=/*
=COPY \5:9990.DEVICE.IOS.RO,\1:\2;YC
=COPY \5:9990.DEVICE.FHS.RO,\1:\2;YC
=COPY \5:9990.FILE.IOS.RO,\1:\2;YC
=COPY \5:9990.FILE.FHS.RO,\1:\2;YC
=COPY \5:9990.GET.TASKID.AG,\1:\2;YC

=/*
=/*      The IOI module was modified to start the TDTI task prior to
=/*      terminating.
=/*
=COPY \5:9990.&.IOI.RO,\1:\2.&.IOI.RO;YC

=/ENDIF

=/*
=/*      Perform the TDTI SYSGEN
=/*
=/@ \1:\2.STD.SYSGEN.CF

=/*
=/*      Establish the original chainfile arguments established when
=/*      TDTI.COPY.CF was executed
=/*
=/@ &.RESTOREA.CF

=/*
=/*      Build the <system>.VERSADOS.MX module to be downloaded
=/*
=DEL \3.VERSADOS.MX
=BUILDS \3.VERSADOS.SY,\3.VERSADOS.MX

=/*
=/*      Two files that are produced as a result of this chainfile are
=/*      &.XTDTIVU.CI and TDTI.LO. If the user has non-standard user-specified
=/*      code that references routines in the TDTI module, these files can
=/*      be of value in reducing the amount of user-specified code required.
=/*      Reference files TTSUCEHL.CD, TE01USC.CI, and TE01USC.LG as an example
=/*      of how these files have been used.
=/*

=END
```

C.4 TDTI.APLICATN.CF

```
=OPTION K
=/*
=/*      OPTION K means:
=/*              Do not translate LOWER case to UPPER case
=/*
=/*
=/*      This chainfile contains the user application related functions.
=/*
=/*      CHAIN INVOCATION
=/*
=/*      This chainfile is called by the example chainfile and as such
=/*      the existing arguments defined by the example chainfile are
=/*      applicable here.
=/*
=/*      Chainfile arguments
=/*
=/@ &.RESTOREA.CF

=/*
=/*      Copy the application tasks to be processed
=/*
=COPY \5:9990.*.TSK*.*,\1:\2.*.TSK*.*;Y
A
=/*
=/*      Assemble the tasks to be processed
=/*
=NOARG
=/@ TSK1.AF
=NOARG
=/@ TSK2.AF
=NOARG
=/@ TSK3.AF
=NOARG
=/@ TSK4.AF
=NOARG
=/@ TSK2TSK3.AF
=/*
=/*      Link the tasks to be processed
=/*
=NOARG
=/@ TSK1.LF
=NOARG
=/@ TSK2.LF
=NOARG
=/@ TSK3.LF
=NOARG
=/@ TSK4.LF
=/*
=/*      Build the corresponding task files to be downloaded
=/*
=/@ TSK1.CF
=/@ TSK2.CF
=/@ TSK3.CF
=/@ TSK4.CF

=END
```

C.5 TDTITTS.EXAMPLE.CF

```
=OPTION K
=/*
=/*      OPTION K means:
=/*                Do not translate LOWER case to UPPER case
=/*
=/*
=/*      This chainfile processes the Task Table, error handling logic,
=/*      and non-standard user specified code.
=/*
=/*      CHAIN INVOCATION
=/*
=/*      This chainfile is called by the example chainfile and as such
=/*      the existing arguments defined by the example chainfile are
=/*      applicable here.
=/*
=/*      Chainfile arguments
=/*
=/@ &.RESTOREA.CF

=/*
=/*      Copy the
=/*                User-specified code that may exist to the target
=/*                     SYSGEN user number
=/*
=/*
=COPY \5:9998.TEMPLATE.TDTITTS.AI,\1:\2;YC
=COPY \5:9990.*.TE05*.*,\1:\2.*.TE05*.*;Y
A
=COPY \5:9990.*.TE01*.*,\1:\2.*.TE01*.*;Y
A

=/*
=/*      Copy the
=/*                Error Handling Logic modules to the target
=/*                     SYSGEN user number
=/*
=/*
=COPY \5:9990.&.TTSEHL.*,\1:\2.&.TTSEHL.*;Y
A

=/*
=/*      Copy the
=/*                Task Table Structure modules to the target
=/*                     SYSGEN user number
=/*
=/* >>>>>
=/* >>>>> The user should examine Task Table Structure (TTS) entries 2 and
=/* >>>>> 5. Both of these entries have user-specified code yet illustrate
=/* >>>>> different types of usage for user-specified code.
=/* >>>>>
=/*
=COPY \5:9990.*.TTS*.*,\1:\2.*.TTS*.*;Y
A

=/*
=/*      Copy the file to initiate a SYSGEN to accumulate the following
=/*      Task Table Structure          Required
=/*      Non-Standard User Code        Optional
=/*      Error Handling Logic          Optional
=/*
=COPY \5:9990.TTSUCEHL.SYSGEN.CF,\1:\2;YC

=/*
=/*      Assemble the user-specified code
=/*
=NOARG
=/@ &.TE05USC.AF
=NOARG
=/@ &.TE01USC.AF

=/*
=/*      Assemble the Error Handling Logic module
=/*
=NOARG
=/@ TTSEHL.AF

=/*
=/*      Assemble the Task Table Structure
=/*
=NOARG
=/@ TTS.AF

=/*
=/*      Restore the arguments established by TDTI.COPY.CF
=/*
=/@ &.RESTOREA.CF

=/*
=/*      Establish volume:user number.catalog as \1:\2.&
=/*
=USE \1:\2.&

=/*
=/*      Execute a SYSGEN to create the module containing
=/*
```

```
=/*      Task Table Structure         Required
=/*      User Specified Code          Optional
=/*      Error Handling Logic         Optional
=/*
=/@ TTSUCEHL.SYSGEN.CF
=/*
=/*      Build the corresponding user-specified code to be downloaded
=/*
=/@ TTSUCEHL.CF

=END
```

THIS PAGE INTENTIONALLY LEFT BLANK.

APPENDIX D

ACCESSIBLE TDTI ROUTINES

D.1 TI70200 -- CHARACTER FILL

```
******************************************************************************
***     INTERNAL SUBROUTINE: TI70200                                       ***
***                                                                        ***
***     DESCRIPTION:                                                       ***
***                                                                        ***
***     Subroutine to character fill a specified area.                     ***
***                                                                        ***
***                                                                        ***
***     NOTES:                                                             ***
***               N/A                                                      ***
***                                                                        ***
***     REGISTER USAGE (A)rgument (D)estroyed (P)reserved (R)eturned       ***
***                                                                        ***
***               0   1   2   3   4   5   6   7   SR hi   SR lo (CCR)      ***
***                                                                        ***
***     D:        AR  P   P   P   P   P   P   P                            ***
***     A:        AR  P   P   P   P   P   P   P                            ***
***                                                                        ***
***                                                                        ***
***     INPUT:                                                             ***
***               A0 = Start address of area to character fill             ***
***                                                                        ***
***               D0 = Low order byte of low order word contains fill      ***
***                         character                                      ***
***                    High order byte of low order word is for future     ***
***                         use                                            ***
***                    High order word contains number of bytes to move    ***
***                                                                        ***
***                    +----------+----------+----------+----------+       ***
***                    |Number of bytes to   | Future   | Fill     |       ***
***                    |character fill       | use      | character|       ***
***                    +----------+----------+----------+----------+       ***
***                                                                        ***
***     OUTPUT:                                                            ***
***                                                                        ***
***     REGISTERS/DATA STRUCTURES                                          ***
***                                                                        ***
***               A0 = Unchanged                                           ***
***               D0 = Unchanged                                           ***
***                                                                        ***
***     CONDITION CODES                                                    ***
***               NOT-EQUAL --> Number of bytes to character fill is       ***
***                                  zero or negative                      ***
***                   EQUAL --> Specified data area was character filled   ***
***                                                                        ***
******************************************************************************
```

D.2 TI70500 -- CREATE TCB

```
********************************************************************************
***     INTERNAL SUBROUTINE: TI70500                                         ***
***                                                                          ***
***     DESCRIPTION:                                                         ***
***                                                                          ***
***     Subroutine to create Task Control Block for specified task.          ***
***                                                                          ***
***                                                                          ***
***     NOTES:                                                               ***
***               If an error occurs, the error handling logic specified     ***
***               in the TASK TABLE structure will be executed via a         ***
***               Jump-to-Subroutine call. The caller is responsible for     ***
***               maintaining the stack pointer so the TASK INITIATOR        ***
***               task can commence normal processing with the next          ***
***               TASK TABLE entry. All other registers will be preserved    ***
***               by the TASK INITIATOR task. If the caller has elected      ***
***               to bypass the error handling logic, processing on this     ***
***               TASK TABLE entry will terminate, and a return to the       ***
***               caller will be executed.                                   ***
***                                                                          ***
***     REGISTER USAGE (A)rgument (D)estroyed (P)reserved (R)eturned         ***
***                                                                          ***
***               0   1   2   3   4   5   6   7   SR hi   SR lo (CCR)        ***
***                                                                          ***
***     D:        P   P   P   P   P   P   P   P                              ***
***     A:        P   P   P   AR  AR  AR  P   P                              ***
***                                                                          ***
***                                                                          ***
***     INPUT:                                                               ***
***               A3 = Address where CREATE TCB parameter block can be       ***
***                         built.                                           ***
***                                                                          ***
***               A4 = Address of TTE to be processed                        ***
***                                                                          ***
***               A5 = TTS start address                                     ***
***                                                                          ***
***     OUTPUT:                                                              ***
***                                                                          ***
***     REGISTERS/DATA STRUCTURES                                            ***
***                                                                          ***
***               A3 = Unchanged                                             ***
***               A4 = Unchanged                                             ***
***               A5 = Unchanged                                             ***
***                                                                          ***
***     CONDITION CODES                                                      ***
***                                                                          ***
***               NOT-EQUAL --> Error occurred processing this TTE           ***
***                                                                          ***
***                   EQUAL --> TTE had TCB created                          ***
***                                                                          ***
********************************************************************************
```

D.3 TI71000 -- CALL EHL

```
*******************************************************************************
***     INTERNAL SUBROUTINE: TI71000                                        ***
***                                                                         ***
***     DESCRIPTION:                                                        ***
***                                                                         ***
***     Subroutine to                                                       ***
***               Call the Error Handling Logic (EHL)                       ***
***                                                                         ***
***     NOTES:                                                              ***
***               The caller is responsible for maintaining the stack       ***
***               pointer so the TASK INITIATOR task can commence normal   ***
***               processing with the next TASK TABLE entry. All other     ***
***               registers will be preserved by the TASK INITIATOR task.  ***
***               If the caller has elected to bypass the error handling   ***
***               logic, processing on this TASK TABLE entry will          ***
***               terminate and a return to the caller will be executed.   ***
***                                                                         ***
***               The user should refer to offset TTSEHLOS for the         ***
***               implementation that will allow the user to bypass the    ***
***               error handling logic                                     ***
***                                                                         ***
***     REGISTER USAGE (A)rgument (D)estroyed (P)reserved (R)eturned        ***
***                                                                         ***
***               0   1   2   3   4   5   6   7   SR hi   SR lo (CCR)       ***
***                                                                         ***
***     D:        AR  P   P   P   P   P   P   P                             ***
***     A:        AR  P   P   P   P   AR  P   P                             ***
***                                                                         ***
***                                                                         ***
***     INPUT:                                                              ***
***                                                                         ***
***               A5 = TASK TABLE structure start address                   ***
***                         use                                             ***
***                                                                         ***
***     OUTPUT:                                                             ***
***                                                                         ***
***     REGISTERS/DATA STRUCTURES                                           ***
***                                                                         ***
***               A5 = Unchanged                                            ***
***                                                                         ***
***     CONDITION CODES                                                     ***
***                                                                         ***
***               N/A                                                       ***
***                                                                         ***
*******************************************************************************
```

D.4 TI71200 -- GET TASK ID

```
******************************************************************************
***     INTERNAL SUBROUTINE: TI71200                                       ***
***                                                                        ***
***     DESCRIPTION:                                                       ***
***                                                                        ***
***     Subroutine to get task identification.                             ***
***                                                                        ***
***                                                                        ***
***     NOTES:                                                             ***
***               N/A                                                      ***
***                                                                        ***
***     REGISTER USAGE (A)rgument (D)estroyed (P)reserved (R)eturned       ***
***                                                                        ***
***               0   1   2   3   4   5   6   7   SR hi   SR lo (CCR)      ***
***                                                                        ***
***     D:        R   P   P   P   P   P   P   P                            ***
***     A:        R   R   P   AR  AR  P   P   P                            ***
***                                                                        ***
***                                                                        ***
***     INPUT:                                                             ***
***                                                                        ***
***               A3 = Address where GET TASK identification parameter     ***
***                         block can be built                             ***
***                                                                        ***
***               A4 = Address of TASK TABLE entry to be processed         ***
***                                                                        ***
***                                                                        ***
***                                                                        ***
***     OUTPUT:                                                            ***
***                                                                        ***
***     REGISTERS/DATA STRUCTURES                                          ***
***                                                                        ***
***            A0/A1 = New 8-byte task identification                      ***
***               A3 = Unchanged                                           ***
***               A4 = Unchanged                                           ***
***                                                                        ***
***               D0 = Results of TRAP request                             ***
***                                                                        ***
***     CONDITION CODES                                                    ***
***                                                                        ***
***               NOT-EQUAL --> Error occurred trying to get task          ***
***                                  identification                        ***
***                                                                        ***
***                   EQUAL --> Task identification successfully obtained  ***
***                                                                        ***
******************************************************************************
```

D.5 TI71300 -- ALLOCATE SECTORS

```
*******************************************************************************
***     INTERNAL SUBROUTINE: TI71300                                        ***
***                                                                         ***
***     DESCRIPTION:                                                        ***
***                                                                         ***
***     Subroutine to allocate task segments as required.                   ***
***                                                                         ***
***     NOTES:                                                              ***
***               If an error occurs, the error handling logic specified   ***
***               in the TASK TABLE structure will be executed via a        ***
***               Jump-to-Subroutine call. The caller is responsible for    ***
***               maintaining the stack pointer so the TASK INITIATOR       ***
***               task can commence normal processing with the next         ***
***               TASK TABLE entry. All other registers will be preserved   ***
***               by the TASK INITIATOR task. If the caller has elected     ***
***               to bypass the error handling logic, processing on this    ***
***               TASK TABLE entry will terminate and a return to the       ***
***               caller will be executed.                                  ***
***                                                                         ***
***     REGISTER USAGE (A)rgument (D)estroyed (P)reserved (R)eturned        ***
***                                                                         ***
***               0   1   2   3   4   5   6   7   SR hi   SR lo (CCR)       ***
***                                                                         ***
***     D:        P   P   P   P   P   P   P   P                             ***
***     A:        P   P   P   AR  AR  AR  P   P                             ***
***                                                                         ***
***     INPUT:                                                              ***
***               A3 = Address where GET SEGMENT parameter block can be     ***
***                         built. The start address of TDTI's dynamic      ***
***                         data area can be used for this address.         ***
***                                                                         ***
***               A4 = Address of TASK TABLE entry to be processed          ***
***                                                                         ***
***               A5 = TASK TABLE structure start address                   ***
***                         use                                             ***
***                                                                         ***
***     OUTPUT:                                                             ***
***                                                                         ***
***     REGISTERS/DATA STRUCTURES                                           ***
***                                                                         ***
***               A3 = Unchanged                                            ***
***               A4 = Unchanged                                            ***
***               A5 = Unchanged                                            ***
***                                                                         ***
***     CONDITION CODES                                                     ***
***                                                                         ***
***               NOT-EQUAL --> Error occurred trying to allocate task      ***
***                                  segment                                ***
***                                                                         ***
***                   EQUAL --> All task segments allocated successfully    ***
***                                                                         ***
*******************************************************************************
```

D

D.6 TI72800 -- START TASK

```
*******************************************************************************
***     INTERNAL SUBROUTINE: TI72800                                        ***
***                                                                         ***
***     DESCRIPTION:                                                        ***
***                                                                         ***
***     Subroutine to start specific task table entry in the TASK          ***
***               TABLE structure.                                          ***
***                                                                         ***
***                                                                         ***
***     NOTES:                                                              ***
***                                                                         ***
***               PRE-REGISTER initialization for a task can ONLY be        ***
***                    accomplished by starting a specific task             ***
***                    from user-specified code.                            ***
***                                                                         ***
***               If pre-register initialization is NOT selected, then      ***
***                    the register values for a task that is started       ***
***                    will be zeros.                                       ***
***                                                                         ***
***               An attempt by the caller to start a task that has         ***
***                    already been started WILL NOT be construed as        ***
***                    an ERROR. The routine will return as if the          ***
***                    request was successful.                              ***
***                                                                         ***
***               If an error occurs, the error handling logic specified    ***
***               in the TASK TABLE structure will be executed via a        ***
***               Jump-to-Subroutine call. The caller is responsible for    ***
***               maintaining the stack pointer so the TASK INITIATOR       ***
***               task can commence normal processing when the return via   ***
***               the RTS instruction is executed. All other registers      ***
***               will be preserved by the TASK INITIATOR task. If the      ***
***               caller has elected to bypass the error handling logic,    ***
***               processing to start this task will terminate and the      ***
***               appropriate status condition will be returned.            ***
***                                                                         ***
***                                                                         ***
***     REGISTER USAGE (A)rgument (D)estroyed (P)reserved (R)eturned        ***
***                                                                         ***
***               0   1   2   3   4   5   6   7   SR hi   SR lo (CCR)       ***
***                                                                         ***
***     D:        P   P   P   P   P   P   P   P                             ***
***     A:        P   P   AR  AR  AR  AR  P   P                             ***
***                                                                         ***
***                                                                         ***
***     INPUT:                                                              ***
***                                                                         ***
***               A2 = If contents of this register equal zero there is     ***
***                         no pre-register initialization                  ***
***                                                                         ***
***                    ELSE                                                 ***
***                                                                         ***
***                    Contents of this address contains the address of    ***
***                         the pre-register initialization data for       ***
***                         registers D0-D7 and A0-A6                      ***
***                                                                        ***
***               A3 = Register contains address where START TASK          ***
***                         parameter block can be built                   ***
***                                                                        ***
***               A4 = Contains address of the specific TTE                ***
***                         to be started                                  ***
***                                                                        ***
***               A5 = Address of TTS                                      ***
***                                                                        ***
***                                                                        ***
***     OUTPUT:                                                            ***
***                                                                        ***
***     REGISTERS/DATA STRUCTURES                                          ***
***                                                                        ***
***               A2 = Unchanged                                           ***
***                                                                        ***
***               A3 = Unchanged                                           ***
***                                                                        ***
***               A4 = Unchanged                                           ***
***                                                                        ***
***               A5 = Unchanged                                           ***
***                                                                        ***
***                                                                        ***
***     CONDITION CODES                                                    ***
***                                                                        ***
***                   ZERO --> Task was started successfully               ***
***                                                                        ***
***               NOT-ZERO --> Error encountered trying to start task      ***
***                            Potential errors are                        ***
***                                 Unable to get task identification      ***
***                                 Unable to start task                   ***
***                                                                        ***
******************************************************************************
```

D

THIS PAGE INTENTIONALLY LEFT BLANK.

APPENDIX E

ERROR INDEX VALUE

E.1 DESCRIPTION

When an error is encountered by the Table Driven Task Initiator (TDTI) task, the user-specified Error Handling Logic (EHL), if it exists, is executed via a Jump-to-Subroutine (JSR) call. To aid in the debugging process, the TDTI task will place on the stack an Error Index Value (EIV). This EIV will be the longword immediately following the return address on the stack. This EIV, when applied to the EIV table, will identify for the user the type of error encountered and the approximate location in the TDTI task where the error occurred. The TDTI task will be responsible for removal of the EIV from the stack in the event the user elects to return from the EHL.

```
+--------------------+
| Return address from|
| EHL                |
+--------------------+
| Error Index Value  |
|                    |
+--------------------+
```

E.2 ERROR INDEX VALUE TABLE

The EIV table has the following EIV code structure:

EIV RANGE	DESCRIPTION
$0000 - $1FFF	TDTI Errors

EIV$0010

General location: TI20000 mainline code.
Problem: After obtaining a data segment, TDTI zeros it out. An error will occur if the number of bytes to zero out is zero or negative.

EIV$0020

General location: TI20000 mainline code.
Problem: TDTI tries to obtain the segment which contains the TT structure, and user-specified code or EHL that may exist, so it can process it. An error will occur if this segment cannot be obtained.

EIV$0030 General location: TI60000 routine.
Problem: The TDTI task will attempt to find a TT entry based on an index value. An error will occur if the TT entry cannot be found.

EIV$0040 General location: TI72800 routine.
Problem: Prior to starting a task, the work area set aside for starting the task is initialized to zeros. An error will occur if the number of bytes to clear is zero or negative.

EIV$1010 General location: TI70500 routine.
Problem: An error occurred when the create TCB request was made.

EIV$1020 General location: TI71300 routine.
Problem: An error occurred because the number of segments to allocate for the task was less than one.

EIV$1030 General location: TI71300 routine.
Problem: An error occurred trying to obtain the task identification.

EIV$1040 General location: TI71300 routine.
Problem: An error occurred trying to allocate a segment for the task.

EIV$1050 General location: TI71300 routine.
Problem: An error occurred trying to declare a segment shareable.

EIV$1054 General location: TI71300 routine.
Problem: An error occurred trying to transfer a segment to the target task.

EIV$1060 General location: TI72800 routine.
Problem: An error occurred trying to obtain the task identification.

EIV$1070 General location: TI72800 routine.
Problem: An error occurred trying to start the task.

APPENDIX F

INTEGRATING A PASCAL TASK INTO A TDTI SYSTEM

F.1 PROBLEM ASSOCIATED WITH PASCAL TASKS IN A TDTI SYSTEM

The example presented in the manual describes how to integrate an assembly language task into the Table-Driven Task Initiator (TDTI) system. The technique used cannot be directly applied to tasks written in Pascal, for two reasons. First, the Pascal runtime initializer expects logical units 5 and 6 to be assigned to a standard input and standard output device, respectively, when the task is first started. Second, the length of the command line is normally passed in a register at startup time. In the case of a TDTI system, there is no command line.

The above functions are normally performed by the VERSAdos session manager when the Pascal task is started in the normal full VERSAdos environment. Since tasks that TDTI starts are already present in memory and not loaded by the session manager, it is necessary to provide for the passing of logical units 5 and 6 and for the correct handling of a nonexistent command line in a different way. One technique for doing this is described below. The technique has been applied to Pascal tasks, but it could also apply to tasks written in other high level languages, or even to tasks written in assembly language if desired.

F.2 RECOMMENDED SOLUTION TO PROBLEM

The recommended solution to the above problem is to provide code to pass logical units 5 and 6, and to indicate an empty command line to the task being started. This code will be pointed to by the "non-standard startup" cell in the associated task entry of TDTI's task startup table. In addition to these functions, the code can initialize the registers of the task being started. This code will have to perform all of the other functions associated with task startup that the TDTI task would perform in the "standard" case. This code consists of a small user-written module and a supplied subroutine module that will perform most of the work.

The non-standard startup module has been reproduced in a later section of this supplement. The following diagram shows the structural relationships between TDTI and the startup module.

```
                                           .
   Task table            TDTI task         .  "Non-standard startup code"
       .                 ---------         .
       .    Non-standard |       |         .  BSR-->
       .    startup cell |       |<------------
   ----------    -------->|       |        .  <--RTS |
   |Entry A |   |         |       |        .         |
   |        |   |         ---------        .         V
   ----------   |                          .  -----------       -----------
   |Entry B |----                          .  | Small    | BSR-->  | Non-     |
   |        |                              .  | user-    |<------->| standard |
   ----------                              .  | written  | <--RTS  | startup  |
   |Entry C |                              .  | interface|         | module   |
   |        |                              .  -----------       -----------
   ----------                              .
       .                                   .
       .                                   .
       .
```

The user-written interface indicated above is very simple. It is illustrated below:

```
         XREF     NSSUPASC
START    MOVE.L   #CN11,A1       Indicate the device mnemonic
*                                desired for the assignment of
*                                logical units 5 and 6.

         BSR      NSSUPASC       Call the non-standard startup
                                 module.

         IF  <NE>  THEN.S
            BRA.S *              Here we hang on a branch if there
*                                was an error in the startup.
         ENDI

         RTS                     Return to TDTI.
```

Note that there is no mention of command line in the above routine. This is because the non-standard startup module will indicate to the Pascal task that the command line length was 0 (no command line). If the user had desired to initialize some register values and have them passed to the task, that could have been done in the above module. The registers that can be passed are A0-A3, and D0-D5. Note that D6 and D7 are set by the startup module to contain command line length (0) and logical unit bit mask ($61) respectively. Registers A4-A6 are reserved and cannot be used to pass information. The terminal identification must be passed in register A1. What is presented above, however, is the minimum necessary to start a Pascal task.

F.3 PASCAL TDTI SYSTEM EXAMPLE

F.3.1 System Description

The TDTI demonstration system is a VM04 which contains RMS, a serial input/output driver, the VM22 driver, the Intelligent Peripheral Controller (IPC) driver, the TRAP #3 server (File Handling Services (FHS)), the I/O subsystem (Input/Output Services (IOS)), and TDTI. Applications include two Pascal tasks, non-standard user-written code for each task, the non-standard startup routine, and the task table created via the TDTI support utility TTGEN.

The Pascal tasks, which use the shareable Pascal runtime library, illustrate the use of the library by de-assigning the standard input/output and then re-assigning them. After this is done, each task goes into an infinite loop displaying a message to "CN11", the terminal chosen for this example.

F.3.2 Memory Map

The INITDAT module, which must be modified to reflect the user's partitioning, contains three partitions:

a. RAM partition 0, which RMS will use to obtain RAM for system as well as user requirements.

b. RAM partition 2, which is defined to contain the read/write segment associated with the Pascal task.

c. ROM partition to contain the following:

   . Pascal task code and shareable runtime library

   . Task table to be processed by TDTI

   . Error handling logic referenced by the task table

   . Non-standard user-written logic referenced by the task table

   . Non-standard startup routine called from the user-written non-standard code

```
00000     +-----------------------------+     ---+
          |                             |        |
          | RAM for TDTI system usage;  |        |---> RAM
          | VERSAdos is resident here   |        |     Partition
          |                             |        |     0
ED000     +-----------------------------+     ---+
          | Read/write segment for      |        |
          | Pascal task A (SEG2)        |        |---> RAM
EE800     +-----------------------------+        |     Partition
          | Read/write segment for      |        |     2
          | Pascal task B (SEG2)        |        |
F0000     +-----------------------------+     ---+
          | Code segment for Pascal     |        |
          | task A        (SEG1)        |        |---> ROM
F0D00     +-----------------------------+        |     Partition
          | Code segment for Pascal     |        |
          | task B        (SEG1)        |        |
F1B00     +-----------------------------+        |
          | Error handling logic        |        |
F1C00     +-----------------------------+        |
          | Non-standard startup        |        |
          | routine                     |        |
F2000     +-----------------------------+        |
          | Non-standard user-written   |        |
          | code for Pascal task A      |        |
F2200     +-----------------------------+        |
          | Non-standard user-written   |        |
          | code for Pascal task B      |        |
F4000     +-----------------------------+        |
          | TDTI task table             |        |
          |                             |        |
F6000     +-----------------------------+        |
          | PASCAL shareable runtime    |        |
          | library        (RRTL)       |        |
FF000     +-----------------------------+        |
          | Available for use           |        |
          |                             |        |
FFFFF     +-----------------------------+     ---+
```

To aid in explaining the memory map shown above, the link files for the Pascal tasks are shown below:

TASK A

```
=/IFC \1
   =ARG TASKA.LL
=/ENDIF
=/*
=LINK ,ATASK.LO,\1;HAMIXS
TASK ATAS,'$0010
PRIORITIES $42,$7F
ATTRIBUTES P
SEG SEG1(R):0,9 $F0000
SEG SEG2:15     $ED000
SEG RRTL(GR):8  $F6000
IN 9998.RRTL.RTLINIT.RO
IN          &.TASKA.RO
IN 9998.RRTL.RRTLACCS.RO
IN 9998.RRTL.PRTL.RO
IN 9998.RRTL.PLJSR.RO
IN 9998.RRTL.RFINIT.RO
IN 9998.RRTL.RTRAPS.RO
IN 9998.RRTL.RPSCALIB.RO
LIB 0.&.PDOLRLIB.RO
END
=/*
=END
```

TASK B

```
=/IFC \1
   =ARG TASKB.LL
=/ENDIF
=/*
=LINK ,BTASK.LO,\1;HAMIXS
TASK BTAS,'$0010
PRIORITIES $42,$7F
ATTRIBUTES P
SEG SEG1(R):0,9 $F0D00
SEG SEG2:15     $EE800
SEG RRTL(GR):8  $F6000
IN 9998.RRTL.RTLINIT.RO
IN          &.TASKB.RO
IN 9998.RRTL.RRTLACCS.RO
IN 9998.RRTL.PRTL.RO
IN 9998.RRTL.PLJSR.RO
IN 9998.RRTL.RFINIT.RO
IN 9998.RRTL.RTRAPS.RO
IN 9998.RRTL.RPSCALIB.RO
LIB 0.&.PDOLRLIB.RO
END
=/*
=END
```

Note that SEG2, which is a read/write segment, has been removed from the ROM partition and placed in partition 2. Partition 2 has been defined with a memory type different than partition 1 so that RMS will not allocate that memory for the system/user memory requirements. Segments SEG1 and RRTL are read-only segments that have been placed in the ROM partition. SEG1 represents the Pascal code for each respective task while RRTL is the shareable runtime library that is common to both Pascal tasks.

The task table, non-standard user-written code, non-standard startup routine, and error handling logic reside in the ROM partition at user-chosen addresses discussed in section S.3.4.

F.3.3 Task Table Generation

The TDTI support utility, TTGEN, will create the major part of the task table required for the TDTI task.

The load modules created by the Pascal link files shown above were used as input to the TTGEN utility to create the task table. Note that in addition to specifying the start address of each segment, the linker commands 'TASK' and 'PRIORITIES' are used to define the taskname, session number, and priorities. This information, if supplied at link time, will automatically be extracted from the Loader Information Block and placed in the task table entry by TTGEN.

The modifications that were made to the task table created by the TTGEN utility were as follows:

- Defining the external references to the mnemonics defined.
- Defining the offset mnemonic to the non-standard user-written routine for each Pascal task.
- Setting a bit (bit 10) in the segment attributes of SEG1 and RRTL to indicate that they reside in a ROM segment.
- Adding an 'END' statement.

The results of running the TTGEN utility with the above modifications follow:

```
          PAGE
* Table Driven Task Initiator (TDTI) startup table.
* Created with TTGEN

          SECTION 4
          XREF    TTSEHL                   (NOTE)
          XREF    NSSUTEOA                 (NOTE)
          XREF    NSSUTEOB                 (NOTE)
*         ORG     not def

*-----------------------------------------------------------------------------*
*            T A S K   T A B L E   H E A D E R                                *
*-----------------------------------------------------------------------------*

          DC.L    '!HDR'           Header ID.

          DC.L    TTSEHL-*         Address offset of error handling routine.
*                                  Zero means no error handling routine.

          DC.L    HEDR_END-*       Address offset to first table entry.

HEDR_END  EQU     *
*-----------------------------------------------------------------------------*
*            E N D   O F   T A S K   T A B L E   H E A D E R                  *
*-----------------------------------------------------------------------------*
```

```
          PAGE
*---------------------------------------------------------------------------*
*                        TASK    ENTRY    TEOA                              *
*---------------------------------------------------------------------------*
          DC.L   'TEOA'             Task entry ID
          DC.L   TEOA_END-*         Offset to next entry
          DC.L   NSSUTEOA-*         Offset to special non-standard processing
                                            (NOTE)                 DC.W     0
        Processing order value
          DC.B   0                  Reserved
          DC.B   'R'                State ('D'=dormant, 'R'=ready)
          DC.L   'ATAS'             Task name
          DC.L   '0010'             '....' Task session number
          DC.W   %0000000000000000  TCB directive options
          DC.L   $0                 Monitor task name
          DC.L   $00000000          '....' Monitor session number
          DC.B   $42                Task initial priority
          DC.B   $7F                Task limit priority
          DC.W   %0000100000000000  Task attributes
          DC.L   $000F0000          Task start address
          DC.W   0                  User-generated task ID
          DC.W   3                  Number of segments for task
*
*------------ Segment entry for task 'TEOA', segment 'SEG2'
*
          DC.W   %0000000100000000  Segment directive options
          DC.W   %0000000000000000  Segment attributes
          DC.L   'SEG2'             Segment name
          DC.L   $000ED000          Start address of segment
          DC.L   $00001400          Length of segment

*
*------------ Segment entry for task 'TEOA', segment 'SEG1'
*
          DC.W   %0000000100000000  Segment directive options
          DC.W   %0100010000000000  Segment attributes    (NOTE)
          DC.L   'SEG1'             Segment name
          DC.L   $000F0000          Start address of segment
          DC.L   $00000D00          Length of segment

*
*------------ Segment entry for task 'TEOA', segment 'RRTL'
*
          DC.W   %0000000100000000  Segment directive options
          DC.W   %0101010000000000  Segment attributes    (NOTE)
          DC.L   'RRTL'             Segment name
          DC.L   $000F6000          Start address of segment
          DC.L   $00008F00          Length of segment

TEOA_END  EQU    *

*--------------------------------------------------------------------------*
*                     END OF ENTRY FOR --- TEOA                            *
*--------------------------------------------------------------------------*
```

```
         PAGE
*-----------------------------------------------------------------------*
*                        TASK    ENTRY    TEOB                          *
*-----------------------------------------------------------------------*
         DC.L   'TEOB'              Task entry ID
         DC.L   0                   Offset to next entry
         DC.L   NSSUTEOB-*          Offset to special non-standard processing
                                            (NOTE)                DC.W    0
       Processing order value
         DC.B   0                   Reserved
         DC.B   'R'                 State ('D'=dormant, 'R'=ready)
         DC.L   'BTAS'              Task name
         DC.L   '0010'              '....' Task session number
         DC.W   %0000000000000000   TCB directive options
         DC.L   $0                  Monitor task name
         DC.L   $00000000           '....' Monitor session number
         DC.B   $42                 Task initial priority
         DC.B   $7F                 Task limit priority
         DC.W   %0000100000000000   Task attributes
         DC.L   $000F0D00           Task start address
         DC.W   0                   User-generated task ID
         DC.W   3                   Number of segments for task
*
*------------ Segment entry for task 'TEOB', segment 'SEG2'
*
         DC.W   %0000000100000000   Segment directive options
         DC.W   %0000000000000000   Segment attributes
         DC.L   'SEG2'              Segment name
         DC.L   $000EE800           Start address of segment
         DC.L   $00001400           Length of segment

*
*------------ Segment entry for task 'TEOB', segment 'SEG1'
*
         DC.W   %0000000100000000   Segment directive options
         DC.W   %0100010000000000   Segment attributes     (NOTE)
         DC.L   'SEG1'              Segment name
         DC.L   $000F0D00           Start address of segment
         DC.L   $00000D00           Length of segment

*
*------------ Segment entry for task 'TEOB', segment 'RRTL'
*
         DC.W   %0000000100000000   Segment directive options
         DC.W   %0101010000000000   Segment attributes     (NOTE)
         DC.L   'RRTL'              Segment name
         DC.L   $000F6000           Start address of segment
         DC.L   $00008F00           Length of segment

TEOB_END  EQU    *

*-----------------------------------------------------------------------*
*                     END OF ENTRY FOR --- TEOB                         *
*-----------------------------------------------------------------------*
```

```
*-----------------------------------------------------------------------------*
*                                                                             *
*             E N D   O F   T A S K   T A B L E                               *
*                                                                             *
*                         Have a nice day !                                   *
*                                                                             *
*-----------------------------------------------------------------------------*
         END
```

NOTE

Lines flagged with (NOTE) have been modified.

F.3.4 SYSGEN Command File

The integration of the task table, non-standard code, and error handling logic into the ROM partition was done via a SYSGEN. The following SYSGEN command file processes, respectively, these items:

. Error handling logic
. Non-standard startup routine
. Non-standard user-written code for each task
. Task table

Note that the SYSGEN 'EXCLUDE' command is used to process the non-standard startup routine as well as the non-standard user-written code for each Pascal task. The link will include the segment specified so that external references can be satisfied, but the SYSGEN 'EXCLUDE' command will remove them from the resultant load module.

The following addresses, excluding those referencing TDTI, can be related to the memory map in section F.3.2.

```
*
*       TTSUCEHL.CD
*
*       This command file is used to accumulate the following modules
*
*               Task Table                      Required
*               Non-standard user code          Optional
*               Error handling logic            Optional
*
*       The addresses should reflect those to be processed by the
*       Table Driven Task Initiator task (TDTI).
*
*       The values defined in this file will be used by the
*       VERSAdos SYSGEN and the task table SYSGEN
*
*
*       Start address of TDTI task
*
TDTISTRT =      $DE00
```

```
*       The following flags are defined as follows
*
*       Non-zero  --> The information         exists
*       Zero      --> The information does not exist
*
FLAGEHL  =         1         Flag for error handling logic
FLAGUSC  =         1         Flag for user-specified code


IFNE    \FLAGEHL

MSG     ***************************************************
MSG     *                                                 *
MSG     * Process the Error Handling Logic                *
MSG     *                                                 *
MSG     ***************************************************

*
*       TTSEHL.CI
*
*       This command file has been built to create a task table
*       error handling logic module.
*
*       STARTEHL = Start address of the error handling logic. This address
*                     is in the range of AREASA to AREAEA defined in module
*                     TDTI.CI
*

PC      =          $F1B00


STARTEHL = *

 PROCESS    &.TTSEHL.LO
 SUBS       &.TTSEHL.LG
 LINK       &.TTSEHL.LG
 END        TTSEHL
 *
 *
 *

ENDC


IFNE    \FLAGUSC

MSG     ***************************************************
MSG     *                                                 *
MSG     * Process User-Specified Code                     *
MSG     *                                                 *
MSG     ***************************************************
```

```
*
*        TDTI.NSSUPASC.CI
*
*        This command file has been built to create the  non-standard startup
*        code routine which can be used with the Table Driven Task Initiator.
*
*        NSSURSA  = Start address for the non-standard startup routine. This
*                        address is in the range of AREASA to AREAEA defined in
*                        module TDTI.CI
*
*


PC = $F1C00

NSSURSA = *

 PROCESS    TDTI.NSSUPASC.LO
 EXCLUDE    TDTI
 SUBS       TDTI.NSSUPAS.LG
 LINK       TDTI.NSSUPAS.LG
 END        NSSUPASC
 *
 *
 *


*
*        NSSU.TASKA.CI
*
*        This command file has been built to create user-written non-standard
*        module for use with the Table Driven Task Initiator task.
*
*        NSSUTASA = Start address of the user-specified code. This address
*                        is in the range of AREASA to AREAEA defined in module
*                        TDTI.CI
*
*


PC = $F2000

NSSUTASA = *

 PROCESS    NSSU.TASKA.LO
 EXCLUDE    TDTI
 EXCLUDE    NSPL
 SUBS       NSSU.TASKA.LG
 LINK       NSSU.TASKA.LG
 END        NSSUTSKA
 *
 *
 *
```

```
*
*       NSSU.TASKB.CI
*
*       This command file has been built to create user-written non-standard
*       module for use with the Table Driven Task Initiator task.
*
*       NSSUTBSA = Start address of the user-specified code. This address
*                  is in the range of AREASA to AREAEA defined in module
*                  TDTI.CI
*
*


PC = $F2200

NSSUTBSA = *

 PROCESS    NSSU.TASKB.LO
 EXCLUDE    TDTI
 EXCLUDE    NSPL
 SUBS       NSSU.TASKB.LG
 LINK       NSSU.TASKB.LG
 END        NSSUTSKB
 *
 *
 *

ENDC


MSG     ***************************************************
MSG     *                                                 *
MSG     * Process the Task Table                          *
MSG     *                                                 *
MSG     ***************************************************

*
*       &.TTS.CI
*
*       This command file has been built to create the task table
*       module for use with the Table Driven Task Initiator task.
*
*       STARTTTS = Start address of the Task Table. This address
*                  is in the range of AREASA to AREAEA defined in module
*                  TDTI.CI
*

PC = $F4000

 STARTTTS = *

 PROCESS    &.TTS.LO
 SUBS       &.TTS.LG
 LINK       &.TTS.LG
 END
 *
 *
 *

         END        SYSGEN
```

F.3.5 Sample Link Files

Sample link files used by the SYSGEN command file for the non-standard start-up routine, non-standard user-written code, and task table are as follows:

a. Non-standard startup routine:

```
=/*      TDTI.NSSUPASC.LG
=/*
=/*      Link file to link the non-standard startup routine
=/*
=LINK ,TDTI.NSSUPASC.LO,TDTI.NSSUPASC.LL;HAMIX
 SEG TDTI:0  $DE00
 SEG NSPL:13 $F1C00
 IN  TTSSA.TDTI.RO
 IN      &.TDTI.RO
 IN   TDTI.NSSUPASC.RO
 END
=END
```

b. Non-standard user-written code for a task:

```
=/*        NSSU.TASKA.LG
=/*
=/*        Link file to link non-standard code for this task entry
=/*
=LINK ,NSSU.TASKA.LO,NSSU.TASKA.LL;HAMIX
 SEG TDTI:0   $DE00
 SEG NSPL:13  $F1C00
 SEG NSPA:2   $F2000
 IN  TTSSA.TDTI.RO
 IN      &.TDTI.RO
 IN   TDTI.NSSUPASC.RO
 IN   NSSU.TASKA.RO
 END
=END
```

c. Task table:

The linker 'DEFINE' commands were used to identify the start address of the error handling logic (TTSEHL), the start address of task 'A' (NSSUTE0A) and task 'B' (NSSUTE0B) non-standard user-written code, respectively.

```
=/*
=/*      TTS.LG
=/*
=/* Link chainfile to create TTS.LO
=/*
=LINK ,&.TTS.LO,TTS.LL;AHMIX
DEFINE   TTSEHL,$F1B00
DEFINE   NSSUTE0A,$F2000
DEFINE   NSSUTE0B,$F2200
SEG TTS0:4 $F4000
INPUT     &.TTS.RO
END
=/*
=END
```

F.3.6 Listing of Non-Standard Startup Routine Code

```
*
*
*                           User-written code
*                                   |
*                                   |
*                                   | Input requirements
*                                   |
*                                   |
*                           Create/start task
*                           in VERSAdos
*                           environment
*                                   |
*                                   |
*                                   |
*                                   |
*                                   |
*                 --------------------------------------
*                 |                                   |
*                 |                                   |
*                 |                                   |
*                 |                                   |
*                 |                                   |
*       Assign standard terminal                 Create task
*       INPUT and OUTPUT
*                 |                                   |
*                 |                                   |
*                 |                                   |
*       ---------------------             --------------------------
*       |                   |             |                       |
*       |                   |             |                       |
*       |                   |             |                       |
*  Assign LUN 5      Assign LUN 6     Transfer LUN          Start task
*                                         |
*                                         |
*                                         |
*                                  ---------------
*                                  |             |
*                                  |             |
*                                  |             |
*                          Transfer LUN 5  Transfer LUN 6
*
```

```
         PAGE
*
*        TDTI.NSSUPASC.SA
*
*        This routine is a non-standard startup routine which executes
*        as a subroutine to the TDTI task via the non-standard user-written
*        code.
*
*        This routine is called to create/start a task by performing the
*        following functions:
*
*                  @ Assign to TDTI, for the user-specified terminal
*                    identification, logical unit 5 for INPUT and logical
*                    unit 6 for OUTPUT
*
*                  @ Create the Task Control Block (TCB) for the task
*
*                  @ Allocate the task's required segments
*
*                  @ Pass logical units 5 and 6 to the task to being processed
*
*                  @ Start the task if requested
*
*        Entry requirements
*
*                  A1 = Initial terminal identification (device mnemonic,
*                       i.e. CN00) the task wants to be assigned to INPUT
*                       and OUTPUT, respectively logical units 5 and 6
*                  A4 = Address of task table entry to process
*                  A5 = Address of task table header
*
*             Registers A4 and A5 are already established upon entry to the
*             user-written code; consequently they need only be passed to this
*             routine.
*
*        Exit status
*
*                  All registers are preserved EXCEPT register D0.
*
*
*        Exit conditions
*
*                  Non-zero --> Error occurred
*                               D0 = Error code in following format
*
*                                   +--------------------+-------------------+
                                    | NSSUPASC INTERNAL  | RMS RETURNED CODE |
*                                   | ERROR CODE         | IF APPLICABLE     |
*                                   |                    | ELSE ZERO         |
*                                   +--------------------+-------------------+
*
```

```
*                   NSSUPASC INTERNAL
*                     ERROR CODES
*
*                          1            Error assigning LUN 5
*                          2            Error assigning LUN 6
*                          3            Error creating Task Control Block
*                          4            Error allocating task segments
*                          5            Error obtaining task identification
*                          6            Error changing LUN 5
*                          7            Error changing LUN 6
*                          8            Error starting task
*
*                       Zero --> Routine completed successfully
*                                DO = ZERO

*
*         EXTERNAL DEFINITIONS
*
          XDEF      NSSUPASC
          XDEF      NSPTASKP


*
*         EXTERNAL REFERENCES
*
*         The method by which the external references, listed below, are
*         satisfied will depend upon the linking mechanism the user selects.
*
*
*         The most common means of satisfying the references are
*
*              1. Via a SYSGEN, to INCLUDE the TDTI segment, which contains
*                 the corresponding external definitions, to satisfy the
*                 external references but EXCLUDE it from the actual linking
*                 of this module, where INCLUDE and EXCLUDE are SYSGEN
*                 commands.
*
*              2. Link this module, using the link DEFINE commands if the user
*                 knows where each external reference resides.
*

*
*                   TDTI routine references
*
          XREF      TI70200   Character fill a specified area
          XREF      TI70500   Create Task Control Block
          XREF      TI71200   Get task identification
          XREF      TI71300   Allocate segments to a task
          XREF      TI72800   Start task

          PAGE
```

```
*
*        INCLUDE   TEMPLATE.TDTITTS.AI
*        INCLUDE     9995.&.EXE.EQ
*        INCLUDE     9995.&.IOE.EQ
*        INCLUDE     9995.&.NIO.EQ
*        INCLUDE     9995.&.FME.EQ
*        INCLUDE     9995.&.FMI.EQ
*
         NOLIST
         INCLUDE   TEMPLATE.TDTITTS.AI
         INCLUDE     9995.&.EXE.EQ
         INCLUDE     9995.&.IOE.EQ
         INCLUDE     9995.&.NIO.EQ
         INCLUDE     9995.&.FME.EQ
         INCLUDE     9995.&.FMI.EQ
         LIST

         PAGE

*
*        MACROS
*
AJDO     MACRO

         SWAP      DO
*                            Exchange high order and low order words
         MOVE.W    D5,DO
*                            Establish the NSSUPASC internal code
         SWAP      DO
*                            Establish DO in error exit format

*                  The exit format for 'DO' is:
*
*                           +--------------------+-------------------+
*                           | NSSUPASC INTERNAL  | RMS RETURNED CODE |
*                           | ERROR CODE         | IF APPLICABLE     |
*                           |                    | ELSE ZERO         |
*                           +--------------------+-------------------+
*

         ENDM

ILLINST  MACRO

         DC.W      $4AFC
*                            Illegal instruction

         ENDM

         PAGE
```

```
*
*       Save registers
*       Assign terminal device, for standard INPUT and OUTPUT,
*          to this routine  (ASTDLUN)
*       IF assignment error
*          Establish error exit status
*          Restore saved registers
*          Return to caller
*       Restore saved registers
*       Create the task for general VERSAdos environment (NSPTASKP)
*       Return to caller
*

        SECTION   13
PBA     EQU       *          Program Base Address for this routine

NSSUPASC:

        MOVE.L    #$61,D7
        CLR.L     D6
*                            Force these registers to conform to the
*                            documentation stated for the Session Control
*                            START command discussed in the VERSAdos
*                            System Facilities manual
*                            D7 = Logical unit assignments for LUN 5 and 6
*                                 and zero for device assignments
*                            D6 = The command line length must be zero


        MOVEM.L   D0-D7/A0-A6,-(A7)
*                            Save the registers

        LEA       PBA(PC),A2
*                            A2 = Program Base Address for this routine
*                                 A2 will maintain this value throughout
*                                 this routine
        BSR.S     ASTDLUN
*                            Assign standard terminal device LUN 5 and 6
*                            for the terminal specified by the user

        IF <NE> THEN.S

            MOVE.L    D0,(A7)
*                            Establish routine exit conditions and register D0
            MOVEM.L   (A7)+,D0-D7/A0-A6
*                            Restore registers that were saved
            RTS
*                            Return to caller

        ENDI

        MOVEM.L   (A7)+,D0-D7/A0-A6
*                            Restore registers that were saved
```

```
         BSR.S     NSPTASKP
*                            Create the task for the general VERSAdos
*                            environment
         RTS
*                            Return to caller

         PAGE

******************************************************************************
***      INTERNAL SUBROUTINE: ASTDLUN                                      ***
***                                                                        ***
***      DESCRIPTION:                                                      ***
***                                                                        ***
***      Subroutine to                                                     ***
***                Establish LUN 5 as the INPUT  device                    ***
***                Establish LUN 6 as the OUTPUT device                    ***
***                                                                        ***
***      NOTES:                                                            ***
***                N/A                                                     ***
***                                                                        ***
***      REGISTER USAGE (A)rgument (D)estroyed (P)reserved (R)eturned      ***
***                                                                        ***
***                0   1   2   3   4   5   6   7   SR hi   SR lo (CCR)     ***
***                                                                        ***
***      D:        DR  P   P   P   P   P   P   P                           ***
***      A:        D   AR  AR  P   AR  AR  P   P                           ***
***                                                                        ***
***                                                                        ***
***      INPUT:                                                            ***
***                                                                        ***
***                                                                        ***
***                A1 = Initial terminal identification the task          ***
***                     wants to be assigned to INPUT and OUTPUT,          ***
***                     respectively logical units 5 and 6.                ***
***                A2 = Program Base Address for this routine              ***
***                A4 = Address of TASK TABLE entry to be processed        ***
***                A5 = Address of TASK TABLE header                       ***
***                                                                        ***
***                                                                        ***
***                                                                        ***
***      OUTPUT:                                                           ***
***                                                                        ***
***      REGISTERS/DATA STRUCTURES                                         ***
***                                                                        ***
***                A1 = Unchanged                                          ***
***                A2 = Unchanged                                          ***
***                A4 = Unchanged                                          ***
***                A5 = Unchanged                                          ***
***                                                                        ***
***                D0 = See CONDITION CODES below for definition           ***
***                                                                        ***
***     CONDITION CODES                                                    ***
***                                                                        ***
***               Not Equal --> Error occurred trying assign LUN           ***
***                                                                        ***
***                   D0 = Error code in following format                  ***
***                                                                        ***
***                        +--------------------+-------------------+      ***
***                        | NSSUPASC INTERNAL  | RMS RETURNED CODE |      ***
***                        | ERROR CODE         | IF APPLICABLE     |      ***
***                        |                    | ELSE ZERO         |      ***
***                        +--------------------+-------------------+      ***
***                                                                        ***
***                                                                        ***
***                   Equal --> LUN assignments were successful            ***
***                                                                        ***
***                   D0 = ZERO                                            ***
***                                                                        ***
******************************************************************************
***                                                                        ***
***     Establish routine input for LUN 5                                  ***
***     Assign LUN 5 for INPUT (NSPLALUN)                                  ***
***     IF error in LUN assignment                                         ***
***        Adjust format of error in D0 to meet routine exit format        ***
***        Establish routine exit conditions                               ***
***        Return to caller                                                ***
***     Establish routine input for LUN 6                                  ***
***     Assign LUN 6 for OUTPUT (NSPLALUN)                                 ***
***     IF error in LUN assignment                                         ***
***        Adjust format of error in D0 to meet routine exit format        ***
***        Establish routine exit conditions                               ***
***        Return to caller                                                ***
***     Return to caller                                                   ***
***                                                                        ***
******************************************************************************

ASTDLUN:

        MOVE.L    #5,D1     LUN
        MOVE.L    #FOPPR,D2 Assign options
*                           A1 = Contains the terminal ID to assign LUN to
*                           D1 = Contains LUN for INPUT assignment
*                           D2 = Contains options for LUN assignment (PR)
*                                Routine input requirements for NSPLALUN
        BSR.S     NSPLALUN
*                           Assign LUN 5 for INPUT
        IF <NE> THEN.S

             MOVE.L    #1,D5
*                           D5 = NSSUPASC internal error code
*                                Error trying to assign logical unit 5
*                           D0 = Results of trap assignment call
*                                Routine input requirements for NSPLAJD0
```

```
             AJD0
*                           Adjust register D0 to reflect the output format
*                           for this routine

             TST.L     D0
*                           Establish routine exit conditions
             RTS
*                           Return to caller
        ENDI

        MOVE.L    #6,D1     LUN
        MOVE.L    #FOPPW,D2 Assign options
                            A1 = Contains the terminal ID to assign LUN to
*                           D1 = Contains LUN for OUTPUT assignment
*                           D2 = Contains options for LUN assignment (PW)
*                                Routine input requirements for NSPLALUN

        BSR.S     NSPLALUN
*                           Assign LUN 6 for OUTPUT

        IF <NE> THEN.S

             MOVE.L     #2,D5
*                           D5 = NSSUPASC internal error code
*                                Error trying to assign logical unit 6
*                           D0 = Results of trap assignment call
*                                Routine input requirements for NSPLAJD0
             AJD0
*                           Adjust register D0 to reflect the output format
*                           for this routine

             TST.L     D0
*                           Establish routine exit conditions

             RTS
*                           Return to caller
        ENDI

        RTS

        PAGE

******************************************************************************
***     INTERNAL SUBROUTINE: NSPLALUN                                      ***
***                                                                        ***
***     DESCRIPTION:                                                       ***
***                                                                        ***
***     Subroutine to                                                      ***
***               Assign the specified device to the specified LUN         ***
***                                                                        ***
***     NOTES:                                                             ***
***               N/A                                                      ***
***                                                                        ***
***     REGISTER USAGE (A)rgument (D)estroyed (P)reserved (R)eturned       ***
***                                                                        ***
***               0   1   2   3   4   5   6   7   SR hi   SR lo (CCR)      ***
***                                                                        ***
***     D:        DR  AR  AR  P   P   P   P   P                            ***
***     A:        P   AR  P   P   P   P   P   P                            ***
***                                                                        ***
***                                                                        ***
***     INPUT:                                                             ***
***                                                                        ***
***                                                                        ***
***               D1 = Contains the LUN to be assigned                     ***
***               D2 = Contains the options for the assignment request     ***
***               A1 = Contains the terminal ID to assign to               ***
***               A2 = Program Base Address for this routine               ***
***                                                                        ***
***                                                                        ***
***     OUTPUT:                                                            ***
***                                                                        ***
***     REGISTERS/DATA STRUCTURES                                          ***
***                                                                        ***
***               D0 = Status of trap call to assign LUN                   ***
***               D1 = Unchanged                                           ***
***               D2 = Unchanged                                           ***
***               A1 = Unchanged                                           ***
***               A2 = Unchanged                                           ***
***                                                                        ***
***     CONDITION CODES                                                    ***
***                                                                        ***
***               Not Equal --> Error occurred trying assign LUN           ***
***                                                                        ***
***                   D0 = Status of trap call to assign LUN               ***
***                                                                        ***
***                                                                        ***
***               Equal     --> LUN assignments were successful            ***
***                                                                        ***
***                   D0 = ZERO                                            ***
***                                                                        ***
******************************************************************************
***                                                                        ***
***     Save registers                                                     ***
***     Put assign parameter block on stack                                ***
***     Establish register input to TI70200 (A0,D0)                        ***
***     Clear the assign parameter block                                   ***
***                                                                        ***
***     IF an error occurs on the parameter block clear                    ***
***     Execute an illegal instruction as this should never occur          ***
***     Build the parameter block on the stack                             ***
***     Initiate the assign call                                           ***
***     Remove the parameter block from the stack                          ***
***     Restore saved registers                                            ***
***     Establish routine exit conditions                                  ***
***                                                                        ***
******************************************************************************
```

```
FHSBLN EQU       $28       Assign LUN parameter block size

NSPLALUN:
        MOVEM.L   A0/A6,-(A7)
*                           Save register
        SUB.L     #FHSBLN,A7
*                           Adjust stack with enough space for parameter
*                           block to assign the terminal to a LUN
        MOVE.L    A7,A0
*                           A0 = Starting address of parameter block
        MOVE.L    #FHSBLN<<16,D0
*                           D0 = Number of bytes to clear in high order word
*                                Fill character in low order byte
        LEA       TI70200-PBA,A6
        LEA       0(A2,A6.L),A6
*                           A6 = Address of routine to execute
        JSR       (A6)
*                           Clear the assign parameter block on the stack
        IF <NE> THEN.S

            ILLINST
*                           Execute an illegal instruction
*                           This should never happen
        ENDI

        MOVE.W    #FHASGN,FHSCMD(A0)
*                           Establish the assign command,code

        MOVE.W    D2,FHSOPT(A0)
*                           Establish assignment options
        MOVE.B    D1,FHSLUN(A0)
*                           Establish the LUN for the terminal device

        MOVE.L    A1,FHSVOL(A0)
*                           Establish the terminal device to assign

        TRAP      #3
*                           Assign the terminal device to the LUN

        ADD.L     #FHSBLN,A7
*                           Remove assign parameter block from stack
        MOVEM.L   (A7)+,A0/A6
*                           Restore register
        TST.L     D0
*                           Establish the routine exit conditions
        RTS

        PAGE
```

```
******************************************************************************
***     INTERNAL SUBROUTINE: NSPTASKP                                      ***
***                                                                        ***
***     DESCRIPTION:                                                       ***
***                                                                        ***
***     Subroutine to                                                      ***
***               Create task in VERSAdos environment by initiating the    ***
***               following:                                               ***
***                                                                        ***
***               Create the Task Control Block (TCB) for task entry       ***
***               Allocate segments associated with task                   ***
***               Pass logical units 5 and 6 to task being created         ***
***               Start task if requested                                  ***
***                                                                        ***
***                                                                        ***
***     NOTES:                                                             ***
***               N/A                                                      ***
***                                                                        ***
***     REGISTER USAGE (A)rgument (D)estroyed (P)reserved (R)eturned       ***
***                                                                        ***
***               0   1   2   3   4   5   6   7   SR hi   SR lo (CCR)      ***
***                                                                        ***
***     D:        DR  P   P   P   P   P   P   P                            ***
***     A:        P   P   P   P   AR  AR  P   P                            ***
***                                                                        ***
***                                                                        ***
***     INPUT:                                                             ***
***                                                                        ***
***               A4 = Address of TASK TABLE entry to be processed         ***
***               A5 = Address of TASK TABLE header                        ***
***                                                                        ***
***               LUN 5 MUST be assigned to the TDTI task prior to call    ***
***               LUN 6 MUST be assigned to the TDTI task prior to call    ***
***                                                                        ***
***     OUTPUT:                                                            ***
***                                                                        ***
***     REGISTERS/DATA STRUCTURES                                          ***
***                                                                        ***
***               A4 = Unchanged                                           ***
***               A5 = Unchanged                                           ***
***                                                                        ***
***               D0 = See CONDITION CODES below for definition            ***
***                                                                        ***
***     CONDITION CODES                                                    ***
***                                                                        ***
***               Not Equal --> Error occurred trying assign LUN           ***
***                                                                        ***
***                   D0 = Error code in following format                  ***
***                                                                        ***
***                        +---------------------+-------------------+      ***
***                        | NSSUPASC INTERNAL   | RMS RETURNED CODE |      ***
***                        | ERROR CODE          | IF APPLICABLE     |      ***
***                        |                     | ELSE ZERO         |      ***
***                        +---------------------+-------------------+      ***
***                                                                        ***
***                                                                        ***
***                   Equal --> LUN assignments were successful            ***
***                                                                        ***
***                   DO = ZERO                                            ***
***                                                                        ***
******************************************************************************
***                                                                        ***
***     Save registers                                                     ***
***     Establish Program Base Address for routine                         ***
***     Establish TDTI work area on stack                                  ***
***     Execute TDTI routine to build TCB                                  ***
***     IF error                                                           ***
***          Remove TDTI work area from stack                              ***
***          Establish error code for TCB build error                      ***
***          Save in register DO                                           ***
***          Restore saved registers                                       ***
***          Exit routine                                                  ***
***     Execute TDTI routine to allocate task segments                     ***
***     IF error                                                           ***
***          Remove TDTI work area from stack                              ***
***          Establish error code for segment allocation                   ***
***          Save in register DO                                           ***
***          Restore saved registers                                       ***
***          Exit routine                                                  ***
***     Remove TDTI work area from stack                                   ***
***     Pass logical unit 5 to task being processed                        ***
***     IF error                                                           ***
***          Adjust to error exit format                                   ***
***          Save in register DO                                           ***
***          Restore saved registers                                       ***
***          Exit routine                                                  ***
***     Pass logical unit 6 to task being processed                        ***
***     IF error                                                           ***
***          Adjust to error exit format                                   ***
***          Save in register DO                                           ***
***          Restore saved registers                                       ***
***          Exit routine                                                  ***
***     Start the task being processed if start requested                  ***
***     IF error                                                           ***
***          Establish error code for trying to start task                 ***
***          Save in register DO                                           ***
***          Restore saved registers                                       ***
***          Exit routine                                                  ***
***     Establish routine exit conditions as successful                    ***
***          Save in register DO                                           ***
***          Restore saved registers                                       ***
***          Exit routine                                                  ***
***                                                                        ***
******************************************************************************
```

```
PBSIZEPL EQU       $5E
*                            Work area for TDTI task
NSPTASKP:

         MOVEM.L   D0-D7/A0-A6,-(A7)
*                            Save registers
         LEA       PBA(PC),A2
*                            A2 = Program Base Address for this routine
         MOVE.L    A7,A6
*                            A6 = Address where registers were saved
         SUB.L     #PBSIZEPL,A7
*                            Adjust stack for TDTI work space
         MOVE.L    A7,A3
*                            A3 = Address of work area that will be used
*                                 by TDTI routines that are called from
*                                 this routine.
         LEA       TI70500-PBA,A1
         LEA       0(A2,A1.L),A1
         JSR       (A1)
*                            Build the TCB for the task being processed

         IF <NE> THEN.S

              ADD.L      #PBSIZEPL,A7
*                            Remove TDTI work area from stack

              MOVE.L     #3<<16,(A7)
*                            D0 = NSSUPASC exit error code
*                                 Error creating TCB
              MOVEM.L   (A7)+,D0-D7/A0-A6
*                            Restore saved registers
              RTS
*                            Return to caller
         ENDI

         LEA       TI71300-PBA,A1
         LEA       0(A2,A1.L),A1
         JSR       (A1)
*                            Allocate the task segments

         IF <NE> THEN.S


              ADD.L      #PBSIZEPL,A7
*                            Remove TDTI work area from stack

              MOVE.L     #4<<16,(A7)
*                            D0 = NSSUPASC exit error code
*                                 Error allocating segments to task
              MOVEM.L   (A7)+,D0-D7/A0-A6
*                            Restore saved registers
              RTS
*                            Return to caller
         ENDI
```

```
        ADD.L     #PBSIZEPL,A7
*                           Remove TDTI work area from stack

        CLR.L     D1
*                           Establish change logical unit options

        MOVE.L    #5,D2
*                           D2 = Logical unit to be passed (LUA)

        MOVE.L    D2,D3
*                           D3 = Logical unit to be received (LUB)

        BSR.S     NSPLCLUN
*                           Pass logical unit 5 to the task being processed

        IF <NE> THEN.S

             IF D5 <EQ> #0 THEN.S

                  MOVE.L    #6,D5
*                           If zero, then error was due to change logical
*                           unit request; else error occurred trying to
*                           obtain task identification, and D5 already has
*                           the internal error code
*                           D5 = NSSUPASC internal error
*                                Error trying to pass logical unit 5
             ENDI


             AJDO
*                           Adjust register D0 to reflect the output format
*                           for this routine

             MOVE.L    D0,(A7)
*                           D0 = NSSUPASC exit error code
*                                Error trying to obtain task identification

             MOVEM.L   (A7)+,D0-D7/A0-A6
*                           Restore saved registers
             RTS
*                           Return to caller

        ENDI


        MOVE.L    #6,D2
*                           D2 = Logical unit to be passed (LUA)

        MOVE.L    D2,D3
*                           D3 = Logical unit to be received (LUB)

        BSR.S     NSPLCLUN
*                           Pass logical unit 6  to the task being processed
```

```
         IF <NE> THEN.S

              IF D5 <EQ> #0 THEN.S

                   MOVE.L    #7,D5
*                            If zero, then error was due to change logical
*                            unit request; else error occurred trying to
*                            obtain task identification, and D5 already has
*                            the internal error code
*                            D5 = NSSUPASC internal error
*                                 Error trying to pass logical unit 6
              ENDI


              AJDO
*                            Adjust register D0 to reflect the output format
*                            for this routine

              MOVE.L    D0,(A7)
*                            D0 = NSSUPASC exit error code
*                                 Error trying to obtain task identification

              MOVEM.L   (A7)+,D0-D7/A0-A6
*                            Restore saved registers
              RTS
*                            Return to caller

         ENDI

         BSR       NSPLSTART
*                            Start the task if task entry so indicates

         IF <NE> THEN.S

               MOVE.L    #8<<16,(A7)
*                            Establish routine exit conditions as unsuccessful
*                            Error trying to start task


               MOVEM.L   (A7)+,D0-D7/A0-A6
*                            Restore saved registers
               RTS
*                            Return to caller
         ENDI

         CLR.L     (A7)
*                            Establish routine exit conditions as successful

         MOVEM.L   (A7)+,D0-D7/A0-A6
*                            Restore saved registers
         RTS
*                            Return to caller

         PAGE
```

```
******************************************************************************
***     INTERNAL SUBROUTINE: NSPLCLUN                                      ***
***                                                                        ***
***     DESCRIPTION:                                                       ***
***                                                                        ***
***     Subroutine to                                                      ***
***               Pass the logical unit to the task being processed        ***
***                                                                        ***
***     NOTES:                                                             ***
***               The SEND option MUST be used. This routine will          ***
***               will force this option.                                  ***
***                                                                        ***
***                                                                        ***
***     REGISTER USAGE (A)rgument (D)estroyed (P)reserved (R)eturned       ***
***                                                                        ***
***               0   1   2   3   4   5   6   7   SR hi   SR lo (CCR)      ***
***                                                                        ***
***     D:        DR  AR  AR  AR  P   DR  P   P                            ***
***     A:        P   P   AR  P   AR  AR  P   P                            ***
***                                                                        ***
***                                                                        ***
***     INPUT:                                                             ***
***                                                                        ***
***               D1 = Options for change logical unit request             ***
***                     SEND option is required                            ***
***               D2 = LUA  the existing assignment                        ***
***               D3 = LUB  the new assignment                             ***
***               A2 = Program Base Address for this routine               ***
***               A4 = Address of TASK TABLE entry to be processed         ***
***               A5 = Address of TASK TABLE header                        ***
***                                                                        ***
***                                                                        ***
***     OUTPUT:                                                            ***
***                                                                        ***
***     REGISTERS/DATA STRUCTURES                                          ***
***                                                                        ***
***               D0 = Trap request status                                 ***
***               D1 = Unchanged                                           ***
***               D2 = Unchanged                                           ***
***               D3 = Unchanged                                           ***
***               D5 = Contains NSSUPASC internal code if the task the     ***
***                    being sent the logical unit does not exist,         ***
***                    else the value will be zero                         ***
***               A2 = Unchanged                                           ***
***               A4 = Unchanged                                           ***
***               A5 = Unchanged                                           ***
***                                                                        ***
***     CONDITION CODES                                                    ***
***                                                                        ***
***               Not Equal --> Error occurred trying assign LUN           ***
***                                                                        ***
***                   Equal --> LUN assignment was successful              ***
***                                                                        ***
******************************************************************************
```

```
***                                                                        ***
***     Save registers                                                     ***
***     Establish work area on stack                                       ***
***     Execute TDTI routine to obtain task identification                 ***
***     IF error                                                           ***
***          Remove work area from stack                                   ***
***          Establish flag - error obtaining task identification          ***
***          Establish error exit conditions                               ***
***          Restore registers                                             ***
***          Return to caller                                              ***
***     Save task identification information                               ***
***     Execute TDTI routine to zero fill parameter work area on stack     ***
***     IF error                                                           ***
***          Execute an illegal instruction as this should not happen      ***
***     Build the change logical unit (pass) parameter block on stack      ***
***     Pass the logical unit to the task being processed                  ***
***     Remove the TDTI work area from the stack                           ***
***     Establish flag; error, if one exists, is due to not obtaining      ***
              task ID                                                      ***
***     Establish the routine exit conditions                              ***
***     Return to the caller                                               ***
***                                                                        ***
******************************************************************************


NSPLCLUN:

         MOVEM.L   A0-A3,-(A7)
*                            Save registers
         SUB.L     #PBSIZEPL,A7
*                            Adjust stack for work area required by TDTI
         MOVE.L    A7,A3
*                            A4 = Address of task table entry being processed
*                            A3 = Address of dynamic work area
*                                 Routine requirement for routines referenced
*                                 in TDTI
         LEA       TI71200-PBA,A1
         LEA       0(A2,A1.L),A1
         JSR       (A1)
*                            Get task ID as TDTI is a real time task
         IF <NE> THEN.S

              ADD.L     #PBSIZEPL,A7
*                            Remove TDTI work area from stack
              MOVE.L    #5,D5
*                            D5 = Contains the NSSUPASC internal code
*                                 Error trying to get task identification
              TST.L     D0
*                            Establish routine exit conditions
              MOVEM.L   (A7)+,A0-A3
*                            Restore registers
              RTS
*                            Return to caller
         ENDI
```

```
        MOVE.L    A1,-(A7)
        MOVE.L    A0,-(A7)
*                           Save respectively session number and task ID
*                           on stack

        MOVE.L    A3,A0
        MOVE.L    #FHSCLN<<16,D0
*                           A0 = Start address of parameter block
*                           D0 = Routine input identifying number of bytes
*                                to fill and the fill character
*                           Routine requirements for TI70200

        LEA       TI70200-PBA,A1
        LEA       0(A2,A1.L),A1
        JSR       (A1)
*                           Zero fill the change LU parameter block
        IF <NE> THEN.S

            ILLINST
*                           Execute an illegal instruction
*                           This should never happen
        ENDI

        MOVE.W    #FHCHLU,(A0)
*                           Establish code, command

        MOVE.W    D1,FHSOPT(A0)
        AND.W     #$FFFE,FHSOPT(A0)
*                           Establish user's specified options
*                           Force the SEND option

        MOVE.L    (A7)+,FHSTSK(A0)
        MOVE.L    (A7)+,FHSSES(A0)
*                           Establish the task name/session number in pb

        MOVE.B    D2,FHSLUA(A0)
        MOVE.B    D3,FHSLUB(A0)
*                           Establish LUA and LUB for the parameter block

        TRAP      #3
*                           Pass the LU to the new task

        ADD.L     #PBSIZEPL,A7
*                           Remove TDTI work area from stack

        CLR.L     D5
*                           Establish flag -- Error, if one exists, is due
*                                to not obtaining task identification
        TST.L     D0
*                           Establish routine exit conditions
        MOVEM.L   (A7)+,A0-A3
*                           Restore registers
        RTS
*                           Return to caller
```

```
        PAGE

******************************************************************************
***     INTERNAL SUBROUTINE: NSPLSTART                                     ***
***                                                                        ***
***     DESCRIPTION:                                                       ***
***                                                                        ***
***     Subroutine to                                                      ***
***               Start task if requested                                  ***
***                                                                        ***
***     NOTES:                                                             ***
***               N/A                                                      ***
***                                                                        ***
***     REGISTER USAGE (A)rgument (D)estroyed (P)reserved (R)eturned       ***
***                                                                        ***
***               0   1   2   3   4   5   6   7   SR hi   SR lo (CCR)      ***
***                                                                        ***
***     D:        P   P   P   P   P   P   P   P                            ***
***     A:        P   P   AR  P   AR  AR  AR  P                            ***
***                                                                        ***
***                                                                        ***
***     INPUT:                                                             ***
***                                                                        ***
***                                                                        ***
***               A2 = Program Base Address for this routine               ***
***               A4 = Address of TASK TABLE entry to be processed         ***
***               A5 = Address of TASK TABLE header                        ***
***               A6 = Address where registers D0-D7/A0-A6 are saved       ***
***                                                                        ***
***                                                                        ***
***                                                                        ***
***     OUTPUT:                                                            ***
***                                                                        ***
***     REGISTERS/DATA STRUCTURES                                          ***
***                                                                        ***
***               A2 = Unchanged                                           ***
***               A4 = Unchanged                                           ***
***               A5 = Unchanged                                           ***
***               A6 = Unchanged                                           ***
***                                                                        ***
***                                                                        ***
***     CONDITION CODES                                                    ***
***                                                                        ***
***               Not Equal --> Error occurred trying to start task        ***
***                                                                        ***
***                   Equal --> Task started successfully                  ***
***                             or did not request starting                ***
***                                                                        ***
******************************************************************************
```

```
***     Save registers                                                       ***
***     If task state is dormant                                             ***
***          Restore saved registers                                         ***
***          Exit task                                                       ***
***     Establish TDTI work area on stack                                    ***
***     Start the requested task                                             ***
***     Remove work area from stack                                          ***
***     IF error                                                             ***
***          Establish error exit conditions                                 ***
***          Restore saved registers                                         ***
***          Return to caller                                                ***
***     Establish successful exit conditions                                 ***
***     Restore saved registers                                              ***
***     Return to caller                                                     ***
***                                                                          ***
********************************************************************************

NSPLSTART:

          MOVEM.L   A1-A3,-(A7)
*                             Save registers

          IF.B #’D’ <NE> TTSINITS+1(A4) THEN.S

               LEA       TI72800-PBA,A1
               LEA       0(A1,A2.L),A1
*                             A1 = Address of routine to execute
               MOVE.L    A6,-(A7)
               MOVE.L    A7,A2
*                             A2 = Contains address where initial registers
*                                  are saved on the stack
               SUB.L     #PBSIZEPL,A7
*                             Adjust stack for TDTI work area
               MOVE.L    A7,A3
*                             A3 = Requirement for TI72800
               JSR       (A1)
*                             Start the requested task
               LEA       PBA(PC),A2
*                             Restore register A2
               ADD.L     #PBSIZEPL,A7
*                             Remove TDTI work area from stack
               MOVEM.L   (A7)+,A6
*                             Restore Register A6

               IF <NE> THEN.S

                    CMP.L     A7,A3
*                             Establish error exit conditions

                    MOVEM.L   (A7)+,A1-A3
*                             Restore save registers
                    RTS
*                             Return to caller
               ENDI
```

```
              CMP.L     D0,D0
*                            Establish good exit conditions

              MOVEM.L   (A7)+,A1-A3
*                            Restore save registers
              RTS
*                            Return to caller

         ELSE.S

              MOVEM.L   (A7)+,A1-A3
*                            Restore save registers
              RTS
*                            Return to caller
         ENDI

         PAGE
         NOP
PRGMSIZE EQU *-NSSUPASC
         END
```

THIS PAGE INTENTIONALLY LEFT BLANK.

INDEX

SUGGESTION/PROBLEM REPORT

MICROSYSTEMS
QUALITY • PEOPLE • PERFORMANCE

Motorola welcomes your comments on its products and publications. Please use this form.

To: Motorola Inc.
Microsystems
2900 S. Diablo Way
Tempe, Arizona 85282
Attention: Publications Manager
Maildrop DW164

Product: ______________________ Manual: ______________________

COMMENTS: ______________________

____

____

____

____

____

____

____

____

____

____

____

Please Print

Name ______________________ Title ______________________

Company ______________________ Division ______________________

Street ______________________ Mail Drop ____________ Phone ____________

City ______________________ State ____________ Zip ____________

For Additional Motorola Publications
Literature Distribution Center
616 West 24th Street
Tempe, AZ 85282
(602) 994-6561

Four Phase/Motorola Customer Support, Tempe Operations
(800) 528-1908
(602) 438-3100

MOTOROLA

MOTOROLA ***Semiconductor Products Inc.***

P.O. BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.

18440 PRINTED IN USA (1/86) MESSENGER 3500