Update Notice #1

February 1987

VAX TDMS
Reference Manual

AD-HU17A-T1

NEW AND CHANGED INFORMATION

This update contains changes and additions made to the *VAX TDMS Reference Manual* for Version 1.7.

INSTRUCTIONS

Place the enclosed pages in the *VAX TDMS Reference Manual* Version 1.7 as replacements for or additions to current pages. Change bars on replacement pages indicate changed text. For new pages and pages where most of the text has been substantially revised, no change bars are used. Instead, only the Version 1.7 release date is shown on the bottom corner of the page.

Old Page	New Page
Title Page/Copyright	Title Page/Copyright
iii to vi	iii to vii
vii to x	ix to xii
xi/xii	xiii/Blank
	1-6.1/Blank
	1-41 to 1-43
2-1 to 2-6	2-1 to 2-6
2-79/2-80	2-79 to 2-80
3-3 to 3-6	3-3 to 3-6
3-13/3-14	3-13 to 3-14
3-31/3-32	3-31/3-32
3-35 to 3-38	3-35 to 3-38
3-41 to 3-46	3-41 to 3-46
3-51 to 3-58	3-51 to 3-58
3-63/3-64	3-63/3-64
3-67/3-68	3-67/3-68
4-15/4-16	4-15/4-16
4-27/4-28	4-27/4-28

Update Notice #1
VAX TDMS
Reference Manual

AD-HU17A-T1

Old Page	New Page
5-15/5-16	5-15/5-16
5-27/5-28	5-27/5-28
Chapter 7	Chapter 7
B-11 to B-14	B-11 to B-14
Index	Index
Reader's Comments/Mailer	Reader's Comments/Mailer

VAX TDMS
Reference Manual

Order No. AA-HU17A-TE
Including AD-HU17A-T1

February 1987

This manual describes the commands, instructions, and synchronous and asynchronous routine calls of VAX TDMS.

OPERATING SYSTEM: **VMS**
MicroVMS

SOFTWARE VERSION: **VAX TDMS V1.7**

digital equipment corporation, maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-paid READER'S COMMENTS form on the last page of this document requests your critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

ACMS	MicroVMS	VAX
CDD	PDP	VAXcluster
DATATRIEVE	Rdb/ELN	VAXinfo
DEC	Rdb/VMS	VAX Information Architecture
DECnet	ReGIS	VIDA
DECUS	TDMS	VMS
MicroVAX	UNIBUS	VT

digital™

Contents

3 Request and Request Library Instructions

How to Use This Manual

This manual describes the commands, instructions, and routine calls for the VAX Terminal Data Management System (VAX TDMS). The VAX TDMS software is also referred to as TDMS in this manual. The VAX DATATRIEVE software is referred to as DATATRIEVE in this manual.

All programming languages referred to in this manual are VAX programming languages.

Intended Audience

This manual is intended for experienced TDMS users who need specific information on a particular command, instruction, or programming call. It is not intended as a learning tool.

If you are new to TDMS, you should read Chapters 1 and 2 of the *VAX TDMS Forms Manual* for an introduction to the product and its components.

Similarly, if you want to learn how to perform a particular task using TDMS, you should read the other manuals in this documentation set:

- For creating forms -- *VAX TDMS Forms Manual*
- For creating requests -- *VAX TDMS Request and Programming Manual*
- For writing application programs -- *VAX TDMS Request and Programming Manual*

Operating System Information

To verify which verions of your operating system are compatible with this version of VAX TDMS, check the most recent copy of the *VAX System Software Order Table/Optional Software Cross Reference Table,* SPD 28.98.xx.

Structure

This manual has eight chapters, four appendixes, and an index:

Chapter 1	Describes the commands for the Form Definition Utility (FDU).
Chapter 2	Describes the commands for the Request Definition Utility (RDU).
Chapter 3	Describes the instructions used for defining requests and request libraries in RDU.
Chapter 4	Describes the synchronous calls used for invoking TDMS from an application program.
Chapter 5	Describes the asynchronous calls used for invoking TDMS from an application program.
Chapter 6	Describes the rules for resolving ambiguous field references in a TDMS request.
Chapter 7	Describes the order in which request instructions are processed at run time.
Chapter 8	Describes the rules for converting data types in TDMS input and output mapping instructions.
Appendix A	Lists FDU error messages, an explanation of the error, and the action the user should take to correct the error.
Appendix B	Lists RDU error messages, an explanation of the error, and the action the user should take to correct the error.
Appendix C	Lists TDMS run-time error messages, an explanation of the error, and the action the user should take to correct the error.
Appendix D	Lists TDMS error message codes that can be issued in a VAX DATATRIEVE application that uses TDMS.

Related Manuals

As you use this book, you may find the following manuals helpful:

VAX TDMS Forms Manual

VAX TDMS Request and Programming Manual

VAX Common Data Dictionary Data Definition Language Reference Manual

VAX Common Data Dictionary Utilities Reference Manual

VAX Run-Time Library Routines Reference Manual

For information on using VAX TDMS with other VAX Information Architecture products, see the *Introduction to Application Development.* This book provides many useful examples and information about creating applications that use VAX TDMS.

Conventions

This section explains the special symbols used in this book:

[ ]	Square brackets in syntax diagrams enclose optional items from which you can choose one or none. Square brackets are also used in Request Definition Utility examples to indicate subscripts in an array.
{ }	Braces enclose items from which you must choose one and only one alternative.
{| |}	Bars in braces indicate that you must choose one or more of the items enclosed.
( )	In RDU syntax, matching parentheses enclose lists of receiving fields in mapping instructions and CDD passwords.
WORD	An uppercase word indicates a command or instruction keyword. Keywords are required unless otherwise indicated. Do not use keywords as variable names.
FDU >	The FDU > prompt indicates the utility is at command level and ready to accept FDU commands.
RDU >	The RDU > prompt indicates the utility is at command level and ready to accept RDU commands.

RDUDFN >	The RDUDFN > prompt indicates that the RDU utility is at the instruction level and ready to accept request or request library instructions.
$	The dollar sign prompt indicates that you are at DIGITAL Command Language (DCL) level and can enter the RDU or FDU utilities. From the DCL prompt, you can also enter RDU or FDU commands if you precede them with the RDU or FDU symbol. (It is possible to change the DCL prompt. However, in this manual the examples use the default prompt, the dollar sign.)
CTRL/x	This key combination indicates that you press both the CTRL (control) key and the specified key simultaneously.
< RET >	This key symbol indicates the RETURN key. Unless otherwise stated, end all example lines by pressing the RETURN key.
Color	Colored text in examples shows what you enter.
...	Horizontal ellipsis means you can repeat the previous item.
.	
.	
.	Vertical ellipsis in an example means that information not directly related to the example has been omitted.

Technical Changes and New Features

This section summarizes the changes to VAX TDMS that are described in this manual:

- Several new commands and instructions have been added to TDMS for this version:

  - ATTACH command (FDU and RDU)
  - DEFINE KEY AS instruction (RDU)
  - SPAWN command (FDU and RDU)

  Descriptions of these commands and instructions are in this manual. In addition, the DEFINE KEY AS instruction is covered in Chapter 11 of the *VAX TDMS Request and Programming Manual.*

- The PROGRAM KEY IS instruction has been extended to support additional keys.
- The %TOD function has been added to the OUTPUT TO and RETURN TO instructions.
- The %MODIFIED function has been added to the RETURN TO instruction.

1.2A ATTACH Command

Transfers control from your process to another process in your job.

Format

```
ATTACH process-name
```

Prompts

```
FDU>
```

```
$
```

Command Parameter

process-name

The name of the process to which control is transferred.

Note

The ATTACH command transfers control from your process to another process in your job. The ATTACH command does not terminate the process from which you issue it. To terminate the process, you can either log out of the process or log out of the parent process.

Example

```
FDU> ATTACH TOMLIN
```

Assume you are in a subprocess and that the parent process name is TOMLIN. This command returns control of your job to the process TOMLIN.

1.22 SPAWN Command

Creates a subprocess of the current process and transfers control of your job to the subprocess.

Format

```
SPAWN [command]

Command Qualifiers                  Defaults

/INPUT=file-spec
/[NO]LOGICAL_NAMES                  /LOGICAL_NAMES
/OUTPUT[=file-spec]                 /OUTPUT=SYS$OUTPUT
/PROCESS=subprocess-name            /PROCESS=subprocess-name
/[NO]SYMBOLS                        /SYMBOLS
/[NO]WAIT                           /WAIT
```

Prompts

```
FDU>
```

```
$
```

Command Parameter

command

The DCL command executed in the subprocess created by the SPAWN command. When the DCL command completes, the subprocess terminates and control is returned to the parent process. If you do not specify a command, TDMS creates a subprocess transferring control to DCL level.

Command Qualifiers

/INPUT=file-spec

Specifies an input file containing one or more DCL command strings to be executed by the spawned subprocess. If you specify a command along with an input file, the command is processed before the commands in the input file. Once processing is complete, the subprocess terminates.

/LOGICAL_NAMES

Specifies that the logical names of the parent process should be copied to the subprocess. The default is /LOGICAL_NAMES.

/NOLOGICAL_NAMES

Specifies that the logical names of the parent process should not be copied to the subprocess. The default is /LOGICAL_NAMES.

/OUTPUT=file-spec

Identifies the file to which the results of the SPAWN operation are written. If you omit the /OUTPUT qualifier, output is written to SYS$OUTPUT.

You should specify output other than SYS$OUTPUT whenever you use the /NOWAIT qualifier. Otherwise, output might be displayed while you are entering new commands.

/PROCESS=subprocess-name

Specifies the name of the subprocess to be created. By default, if you omit the /PROCESS qualifier, a unique process name is assigned with the same user name as the parent process and a unique number. The default subprocess name format is username_n.

/SYMBOLS

Specifies that the DCL global and local symbols should be passed to the subprocess. The default is /SYMBOLS.

/NOSYMBOLS

Specifies that the DCL global and local symbols should not be passed to the subprocess. The default is /SYMBOLS.

/WAIT

Ensures that the system waits until the subprocess is completed before allowing more commands to be issued in the parent process. The default is /WAIT.

/NOWAIT

Allows you to issue new commands while the specified subprocess is running. The default is /WAIT.

When you use the /NOWAIT qualifier interactively, be sure to use the /OUTPUT qualifier as well so that output from the subprocess is directed to a file rather than to your terminal.

Note

If you return to your FDU session by logging out of the subprocess, the subprocess is terminated. To return to your FDU session without terminating the subprocess, use the DCL ATTACH command. You can then return from FDU to the subprocess with the FDU ATTACH command.

Examples

These examples assume that you have defined DMU and RDU as symbols for the commands that invoke these utilities.

```
FDU> SPAWN RDU LIST REQUEST TEST_REQUEST
```

The SPAWN command creates a subprocess that runs RDU to list the request TEST_REQUEST. When RDU completes execution of the LIST command, it returns control to FDU.

```
FDU> SPAWN
$ DMU LIST/TYPE=CDD$RECORD/LIST=LIST_OF_RECORDS.LIS
$ PRINT LIST_OF_RECORDS.LIS
$ LOGOUT
FDU>
```

The SPAWN command creates a DCL subprocess. You can then issue DCL commands. In this example, you list all the records in the CDD default directory and print the listing. Then you log out of the subprocess to return to FDU.

```
FDU> SPAWN/OUTPUT=FORMLISTING.X
$ DIRECTORY _FORM
$ LOGOUT
FDU>
```

The SPAWN command creates a DCL subprocess and directs future output to the file FORMLISTING.X. The DIRECTORY command lists all files that end with "_FORM" in the FORMLISTING.X file. Then you log out of the subprocess to return to FDU.

Request Definition Utility (RDU) Commands 2

This chapter provides complete information for all the commands in the Request Definition Utility (RDU). The command keywords are listed at the top of each page and are in alphabetical order.

Each section contains the following categories, as applicable:

Format	Provides the syntax for the command.
Prompts	Shows the prompts for each command.
Command Parameters	Explains each parameter.
Command Qualifiers	Explains each qualifier and how to use it. Always specify a qualifier following a command and its parameters (at the end of a command line) unless otherwise indicated.
Note	Provides information about using the command.
Examples	Gives examples on using the command.

2.1 Common RDU Qualifier, /AUDIT

Many RDU commands allow you to use the optional qualifier /AUDIT. To avoid repetition, the qualifier is explained fully here and then mentioned in the description of each command that uses it.

The /AUDIT qualifier stores audit text with the request or request library definition. The forms of the qualifier are:

/AUDIT

The standard default audit text includes the date and time you perform the specified operation on the request or request library definition and the name of the utility (RDU). /AUDIT is the default.

/NOAUDIT

Does not store audit text with the request or request library definition. /AUDIT is the default.

/AUDIT = audit-string

Stores, with the request or request library definition, an audit string that consists of one or more single words, one or more quoted strings, text from a file, or a combination of these three items. The optional audit string can indicate, among other things, when the request or request library definition is created, accessed, or changed.

Each item in the audit string (and each line of text in a file) creates one line of audit text. If the audit string is longer than one line, you must specify the hyphen (-) continuation character as the last character on each line you are continuing. When you include more than one item, enclose the list of items in parentheses.

If you specify more than 64 lines of audit text, RDU issues a warning message and truncates the audit text to 64 lines.

/AUDIT is the default.

/AUDIT = single-word

Stores a single word with the request or request library definition. The word need not be enclosed in quotation marks. If you specify a series of single words, enclose the words in parentheses and separate them with commas, for example, /AUDIT = (WORD1, WORD2, WORD3).

/AUDIT = quoted-string

Stores the string with the request or request library definition. The string can be a single line of text between quotation marks. If you specify a series (up to 64 lines) of quoted strings, enclose the strings in parentheses and separate them by commas, for example, /AUDIT = ("first string","second string","third string").

/AUDIT = @file-spec

Stores, with the request or request library definition, the text from the specified file or files. If you specify more than one file, enclose each @file-spec parameter in parentheses and separate by commas. The audit text in the files need not be enclosed in quotation marks. You can specify a total of up to 64 lines of text.

Use the standard VMS file specification. The default file type is .DAT.

/AUDIT={ | single-word, quoted-string, @file-spec, ... | }

Stores, with the request or request library definition, a combination of one or more of the following items: a single word, text from a file, or a quoted string. The list of items must be enclosed in parentheses and separated by commas.

RDU stores up to 64 lines of audit text. Each item (and each line of text in a file) creates one line in the audit text.

2.1A Validate Mode and Store Mode

[No]Validate mode

You use the SET VALIDATE command to set RDU to Validate mode. You use the SET NOVALIDATE command to set RDU to Novalidate mode. Validate mode is the default.

With Validate mode set, when you use the CREATE, MODIFY, or REPLACE command, RDU checks requests and request library definitions for valid references to form definitions and record definitions, and checks request library definitions for requests. RDU also validates form fields and record fields in mappings. If the requests or request library definitions are valid, RDU stores the new requests or request library definitions in the CDD.

In Novalidate mode, RDU does not check requests and request library definitions for valid references. When RDU is set to Novalidate mode, you can create a request before creating the form and/or record(s).

[No]Store mode

Store mode causes certain RDU commands to store the request binary structure along with the request in the CDD. Nostore mode means that the request binary structure is not stored in the CDD with the request.

If the request binary structure is stored with the request, the BUILD LIBRARY command revalidates the request only when an associated form or record has changed since the request binary structure was created. Otherwise, in Nostore mode, the request is revalidated every time you issue a BUILD LIBRARY command.

You can set RDU to Store mode by specifying the /STORE qualifier with the CREATE REQUEST, MODIFY REQUEST, REPLACE REQUEST, VALIDATE LIBRARY, and VALIDATE REQUEST commands. To set RDU to Nostore mode, you specify the /NOSTORE qualifier with those commands.

When RDU is set to Validate mode, Store mode is the default. However, when RDU is set to Novalidate mode, Nostore is the default.

2.2 @file-spec Command

Executes the specified indirect command file that contains RDU commands and associated request or request library definition instructions.

Format

```
@file-spec
```

Prompts

```
RDU>
```

```
$
```

Command Parameter

file-spec

The name of a command file for RDU to execute. Use the standard VMS file specification format. If you do not specify a file type, RDU looks for a file with a .COM file type, which is the default.

Notes

The file can contain commands to process a request or request library definition (CREATE, REPLACE, COPY, MODIFY) as well as other RDU commands.

When RDU executes an indirect command file, it displays any output on SYS$OUTPUT. RDU also displays error messages on SYS$ERROR if SYS$ERROR is different from SYS$OUTPUT.

RDU does not display the RDU commands it is executing from a command file unless the RDU command SET VERIFY is in effect.

Note

When you start RDU, it executes a command file pointed to by the logical name RDUINI (if such a file is present in your current default VMS directory).

By default, the logical name RDUINI points to the command file named RDUINI.COM. You create this file and can place in it startup commands that you wish RDU to execute each time you call the utility. You can define RDUINI to point to any file you wish. If you name the file something other than RDUINI.COM, define the logical name RDUINI to point to the new file.

Examples

```
RDU> @ACCTAPP
```

RDU executes the file ACCTAPP.COM, which can contain, for instance, the commands and request text to create several requests associated with a TDMS accounting application.

```
$ RDU @ACCTAPP
```

You can type the @file-spec command at DCL level.

```
$ RDU
  RDU>
```

RDU automatically executes RDUINI.COM if it is present in your current default directory. The file may contain commands such as:

SET NOVALIDATE
SET LOG
SET VERIFY

2.2A ATTACH Command

Transfers control from your process to another process in your job.

Format

```
ATTACH process-name
```

Prompts

```
RDU>
```

```
$
```

Command Parameter

process-name

The name of the process to which control is transferred.

Note

The ATTACH command transfers control from your process to another process in your job. The ATTACH command does not terminate the process from which you issue it. To terminate the process, you can either log out of the process or log out of the parent process.

Example

```
RDU> ATTACH TOMLIN
```

Assume you are in a subprocess and that the parent process name is TOMLIN. This command returns control of your job to the process TOMLIN.

2.3 BUILD LIBRARY Command

Creates a request library file that contains the requests and the form and record information necessary to execute these requests.

Format

```
BUILD LIBRARY request-library-path-name [request-library-file]
```

Command Qualifiers	*Defaults*
/[NO]AUDIT	/AUDIT
/AUDIT = audit-string	/AUDIT
/[NO]LIST	/NOLIST
/LIST = file-spec	/NOLIST
/[NO]LOG	/NOLOG
/[NO]PRINT	/NOPRINT

Prompts

```
RDU>
```

```
$
```

Command Parameters

request-library-path-name

The CDD path name (given, relative, or full) of the request library definition that contains the names of the requests to be included in the request library file.

request-library-file

The VMS file that RDU builds to contain the requests and the form and record information necessary to execute these requests. Use the standard VMS file specification format. If you assign no file type, RDU supplies the .RLB file type.

2.29 SHOW VERSION Command

Displays information about the current version of RDU to SYS$OUTPUT.

Format

```
SHOW VERSION
```

Prompt

```
RDU>
```

Example

```
RDU> SHOW VERSION
VAX RDU V1.6-0
```

RDU shows the version of the utility you are running.

2.29A SPAWN Command

Creates a subprocess of the current process and transfers control of your job to the subprocess.

Format

SPAWN [command]

Command Qualifiers	*Defaults*
/INPUT=file-spec	
/[NO]LOGICAL_NAMES	/LOGICAL_NAMES
/OUTPUT[=file-spec]	/OUTPUT=SYS$OUTPUT
/PROCESS=subprocess-name	/PROCESS=subprocess-name
/[NO]SYMBOLS	/SYMBOLS
/[NO]WAIT	/WAIT

Prompts

```
RDU>
```

```
$
```

Command Parameter

command

The DCL command executed in the subprocess created by the SPAWN command. When the DCL command completes, the subprocess terminates and control is returned to the parent process. If you do not specify a command, TDMS creates a subprocess transferring control to DCL level.

Command Qualifiers

/INPUT=file-spec

Specifies an input file containing one or more DCL command strings to be executed by the spawned subprocess. If you specify a command along with an input file, the command is processed before the commands in the input file. Once processing is complete, the subprocess terminates.

/LOGICAL_NAMES

Specifies that the logical names of the parent process should be copied to the subprocess. The default is /LOGICAL_NAMES.

/NOLOGICAL_NAMES

Specifies that the logical names of the parent process should not be copied to the subprocess. The default is /LOGICAL_NAMES.

/OUTPUT=file-spec

Identifies the file to which the results of the SPAWN operation are written. If you omit the /OUTPUT qualifier, output is written to SYS$OUTPUT.

You should specify output other than SYS$OUTPUT whenever you use the /NOWAIT qualifier. Otherwise, output might be displayed while you are entering new commands.

/PROCESS=subprocess-name

Specifies the name of the subprocess to be created. By default, if you omit the /PROCESS qualifier, a unique process name is assigned with the same user name as the parent process and a unique number. The default subprocess name format is username_n.

/SYMBOLS

Specifies that the DCL global and local symbols should be passed to the subprocess. The default is /SYMBOLS.

/NOSYMBOLS

Specifies that the DCL global and local symbols should not be passed to the subprocess. The default is /SYMBOLS.

/WAIT

Ensures that the system waits until the subprocess is completed before allowing more commands to be issued in the parent process. The default is /WAIT.

/NOWAIT

Allows you to issue new commands while the specified subprocess is running. The default is /WAIT.

When you use the /NOWAIT qualifier interactively, be sure to use the /OUTPUT qualifier as well so that output from the subprocess is directed to a file rather than to your terminal.

Note

If you return to your RDU session by logging out of the subprocess, the subprocess is terminated. To return to your RDU session without terminating the subprocess, use the DCL ATTACH command. You can then return from RDU to the subprocess with the RDU ATTACH command.

Examples

These examples assume that you have defined FDU as a symbol for the command that invokes this utility:

```
RDU> SPAWN FDU LIST FORM TEST_FORM
```

The SPAWN command creates a subprocess that runs FDU to list the form TEST_FORM. When FDU completes execution of the LIST command, it returns control to RDU.

```
RDU> SPAWN FDU LIST FORM TEST_FORM/OUTPUT=TEST_FORM.LIST/NOWAIT
```

This SPAWN command performs the same function as the previous example. However, the output from the FDU LIST command is sent to the file TEST_FORM.LIST, rather than to the terminal. The /NOWAIT qualifier allows the user to continue working in RDU while the subprocess is creating the TEST_FORM.LIST file.

2.30 VALIDATE LIBRARY Command

Determines whether a request library definition in the CDD is valid. If RDU is in Store mode, the VALIDATE LIBRARY command also creates a request binary structure in the CDD for each request in the request library definition.

Format

```
VALIDATE LIBRARY request-library-path-name

Command Qualifiers                  Defaults

/[NO]AUDIT                          /AUDIT
/AUDIT=audit-string                 /AUDIT

/[NO]LOG                            /NOLOG

/[NO]STORE                          /STORE
```

Prompts

```
RDU>

$
```

Command Parameter

request-library-path-name

The CDD path name (given, relative, or full) of the request library definition that you want to validate.

Command Qualifiers

/AUDIT

Stores audit text with the request library definition. The standard default audit text includes the date and time you validate the request library definition and the name of the utility (RDU). /AUDIT is the default.

If a field has been defined with input highlighting in FDU, the [NO] BLINK instruction will override the input highlighting video attribute for that field. Any other video attributes assigned to the field will be unaffected by the [NO] BLINK instruction.

The BLINK instruction is ignored if you run a TDMS application on a VT52 terminal.

Examples

```
RDUDFN> NO BLINK FIELD DEPARTMENT;
```

Sets the form field DEPARTMENT to no blinking.

```
RDUDFN> BLINK FIELD NAME, BADGE, SEX;
```

Sets the form fields NAME, BADGE, and SEX to blinking.

```
RDUDFN> BLINK FIELD %ALL;
```

Sets all the fields on the active form to blinking.

3.2 [NO] BOLD FIELD Instruction

Sets or clears the bolding video attribute of a field in an active form.

Format

```
[NO] BOLD FIELD { form-field[,...] } ;
                { %ALL             }
```

Prompt

```
RDUDFN>
```

Instruction Parameters

form-field

The name assigned to the form field. The field must be on the active form. You can specify one form field or a list of form fields separated by commas.

%ALL

All the fields on the active form.

Notes

If you specify the BOLD or NO BOLD instruction in a request, it overrides:

- A Bold or No Bold attribute assigned in a form definition.
- A BOLD or NO BOLD instruction that is still active from a previous request call. A video instruction is still active when:
  - A form is still on the screen from a previous request call
  - The current call to a request uses that same form with a USE FORM instruction

At run time, a BOLD or NO BOLD instruction used within a conditional instruction supersedes one in a base request or any outer conditional instruction.

If a field has been defined with input highlighting in FDU, the [NO] BOLD instruction will override the input highlighting video attribute for that field. Any other video attributes assigned to the field will be unaffected by the [NO] BOLD instruction.

The BOLD instruction is ignored if you run a TDMS application on a VT52 terminal.

Examples

```
RDUDFN> BOLD FIELD NAME, BADGE, SEX;
```

Bolds the form fields NAME, BADGE, and SEX.

```
RDUDFN> NO BOLD FIELD NAME;
```

Clears the Bold attribute for the form field NAME.

```
RDUDFN> BOLD FIELD %ALL;
```

Bolds all the fields on the active form.

3.3 [NO] CLEAR SCREEN Instruction

Clears or does not clear the terminal screen before displaying a form.

Format

```
[NO] CLEAR SCREEN;
```

Prompt

```
RDUDFN>
```

Notes

The CLEAR SCREEN instruction ensures that the screen is clear of system messages or other information before TDMS displays a form on the screen. NO CLEAR SCREEN is the default.

TDMS executes a CLEAR SCREEN or NO CLEAR SCREEN instruction before executing any form usage or mapping instructions.

You might want to use this instruction at the beginning of every request, before TDMS displays a form specified in the USE FORM or DISPLAY FORM instruction. Note that the CLEAR SCREEN instruction repaints the entire screen and can be very slow.

At run time, a CLEAR SCREEN or NO CLEAR SCREEN instruction used within a conditional instruction supersedes one in a base request or any outer conditional instruction.

Examples

```
RDUDFN> CLEAR SCREEN;
```

Clears the terminal screen.

```
RDUDFN> NO CLEAR SCREEN;
```

Does not clear the terminal screen.

Examples

```
RDUDFN> USE FORM EMPLOYEE_FORM;
RDUDFN> DEFAULT FIELD EMPLOYEE;
```

Displays EMPLOYEE_FORM with the contents from the immediately previous request call, except for the field EMPLOYEE. Resets the contents of the form field EMPLOYEE to display the contents in the form definition for the field EMPLOYEE.

```
RDUDFN> NO DEFAULT FIELD EMPLOYEE, BADGE, DEPT;
```

Specifies that the contents of the form fields EMPLOYEE, BADGE, and DEPT not be the defaults specified in the form definition.

3.5A DEFINE KEY AS Instruction

Specifies an alternate definition for a key or key sequence.

Format

```
DEFINE KEY key-name AS key-function;
```

Prompt

```
RDUDFN>
```

Instruction Parameters

key-name

The name of the key you want to define to perform the key-function you are specifying. There are several categories of key names.

- The KEYPAD keys are those on the numeric keypad at the right edge of the keyboard. There is a numeric keypad on VT100- and VT200-series terminals. Remember that the KEYPAD keyword is *not* enclosed in quotation marks but the remaining part of the key-name is.

  Note that the keypad must be set to Application mode when KEYPAD keys are used in a request. You use the KEYPAD MODE IS instruction to set the keypad to Application mode.

  The following numeric keypad key names can be specified with the KEYPAD keyword:

0	4	8
1	5	9
2	6	. (period)
3	7	, (comma)
		- (hyphen)

- PF keys are located on the numeric keypad on VT100- and VT200-series terminals. You do not include the KEYPAD keyword as part of the key-name parameter. The PF keys that you can specify are:

PF1	PF3
PF2	PF4

- The F (function) keys are located across the top row of VT200-series keyboards. Keys F1 through F5 are local function keys that cannot be redefined. You can specify the other F keys with the DEFINE KEY AS instruction. When specifying an F key name, do not separate the F from the digit. You can specify the following F keys:

F6	F10	F14	F18
F7	F11	F15	F19
F8	F12	F16	F20
F9	F13	F17	

- Only VT200-series terminals have E keys. These six keys are located on the "editing" keypad, above the arrow keys. The E keys that you can specify are:

E1	E4
E2	E5
E3	E6

- You use the keywords listed below to specify the arrow keys. In addition, you can specify the GOLD keyword with arrow key names.

DOWNARROW	GOLD DOWNARROW
LEFTARROW	GOLD LEFTARROW
RIGHTARROW	GOLD RIGHTARROW
UPARROW	GOLD UPARROW

- There are other key names that you can use for the key-name parameter. The keys are:

BACKSPACE (VT100 mode)
ENTER
LINEFEED (VT100 mode)
RETURN
TAB

Note that you should specify BACKSPACE and LINEFEED only for terminals in VT100 mode. When using VT200-mode, you specify the F12 and F13 keys instead of BACKSPACE and LINEFEED as the key-name.

If you plan to redefine the ENTER key, be sure to set the keypad to Application mode. When the keypad is in Numeric mode, the ENTER key has the same definition as the RETURN key. When the keypad is in Application mode, you can define the ENTER key to have a different function from the RETURN key.

key-function

The operation you want the defined key to perform. The key function can be one of the following:

Function	Description
DONE	Completes data entry and exits from the request.
ERASE	Deletes the contents of the current field.
ERROR	Signals the operator that an error has been made and leaves the cursor where it was.
EXIT_SCROLL_DOWN	Moves the cursor out of a scrolled region to the next field.
EXIT_SCROLL_UP	Moves the cursor out of a scrolled region to the previous field.
GOLD	Combines with another key to perform a specific operation.
HARDCOPY	Copies the current state of the active form into the file assigned to the logical TSS$HARDCOPY.
HELP	Provides help text and/or a help form.
LEFT	Moves the cursor one position to the left within the current field.
NEXT	Moves the cursor to the next field.
PREVIOUS	Moves the cursor to the previous field.
REFRESH	Clears and repaints the screen.
RIGHT	Moves the cursor one position to the right within the current field.
SCROLL_DOWN	Moves the cursor to the next line of a scrolled region.
SCROLL_UP	Moves the cursor to the previous line of a scrolled region.

Keys have the following functions by default:

Key Name	Key Function
BACKSPACE	PREVIOUS
DOWNARROW	SCROLL_DOWN
ENTER	DONE
F12	PREVIOUS
F13	ERASE
F15	HELP
GOLD DOWNARROW	EXIT_SCROLL_DOWN
GOLD UPARROW	EXIT_SCROLL_UP
LEFTARROW	LEFT
LINEFEED	ERASE
PF1	GOLD
PF2	HELP
PF4	HARDCOPY
RETURN	DONE
RIGHTARROW	RIGHT
TAB	NEXT
UPARROW	SCROLL_UP

Any key that can be defined, but is not in this list, maps to the ERROR function by default.

Note that CTRL/R and CTRL/W are assigned the REFRESH function by default. You cannot redefine these key sequences, but you can assign the REFRESH function to other keys or key sequences.

Notes

You can specify only one operation per key.

If TDMS cannot move to a field according to the specified operation, TDMS signals the operator and leaves the cursor in the current field.

RDU cannot determine the keypad mode at run time, so RDU does not check the mode when it validates the request. (See the KEYPAD MODE IS instruction.)

When keys or key sequences have more than one definition, you need to be aware that TDMS processes only one of the key definitions. The following are the rules that TDMS uses to resolve multiple definitions.

- If a key or key sequence is defined as an application function key (AFK), only the AFK function is ever executed. TDMS ignores any key definition instructions (PROGRAM KEY IS or DEFINE KEY AS) for that key or key sequence.
- If there is no AFK definition for a key or key sequence, a key definition instruction within a conditional instruction takes precedence and is executed. A key definition instruction in the base request is ignored.
- If no key definition instruction occurs within a conditional instruction, a key definition instruction in the base request is executed.
- If two or more key definition instructions occur in the base request or in conditional instructions defined at the same level within a request, one of the instructions takes precedence and is executed. However, there are no rules to determine which key definition instruction prevails in this instance.

For example, if you redefine BACKSPACE using the DEFINE KEY AS instruction and also have an AFK for CTRL/H, the AFK definition is the one that is processed when the operator presses CTRL/H or BACKSPACE.

Redefining a key with a default function means that the key no longer has that default function. In most instances, you lose access to the original function. When redefining keys, be sure that:

- At least one key is defined as a request termination key
- The operator knows which key it is

Examples

```
RDUDFN> DEFINE KEY TAB AS DONE;
RDUDFN> DEFINE KEY RETURN AS NEXT;
```

Redefines the TAB key as the TDMS DONE function and the RETURN key as the TDMS NEXT function. Now the operator can use the RETURN key to move from one field to the next and the TAB key to indicate that there is no more data to enter.

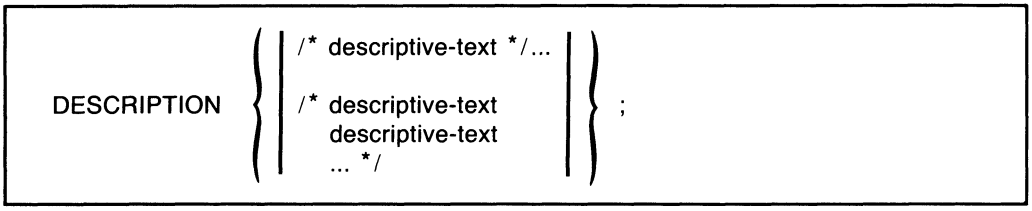
```
RDUDFN> DEFINE KEY F20 AS REFRESH;
```

Redefines the F20 key on a VT200-series terminal to perform the REFRESH function. Now, the operator can press CTRL/R, CTRL/W, or F20 to clear and repaint the screen.

3.6 DESCRIPTION Instruction

Specifies comment text that is stored with the source text of the request in the CDD.

Format



Prompt

```
RDUDFN>
```

Instruction Parameter

descriptive-text

Text you wish to store with the request or request library definition in the CDD. The text must be enclosed between each slash and asterisk combination:

/* and */

Notes

You can use the DESCRIPTION instruction anywhere in the body of the request or request library definition except embedded in a request instruction or a request library definition instruction. The text may describe the purpose of a request or request library definition or some special feature you wish to document.

You *must* use a semicolon (;) at the end of a DESCRIPTION instruction.

The text is printed out if you use the LIST command to list a request or request library definition.

You can also use an exclamation mark (!) anywhere within the request to indicate a comment.

3.13 KEYPAD [MODE] IS Instruction

Specifies whether the terminal keypad is in Application mode or Numeric mode. This instruction is used in conjunction with the DEFINE KEY AS and the PROGRAM KEY IS instructions.

Format

```
KEYPAD [MODE] IS { NUMERIC     } ;
                 { APPLICATION }
```

Prompt

```
RDUDFN>
```

Instruction Parameters

NUMERIC

When you specify the KEYPAD MODE IS NUMERIC instruction in a request, the keypad is set to Numeric. When an operator presses a key on the keypad, an application program receives the data from that key as either digits (0-9) or a punctuation mark (period, comma, and hyphen keys).

APPLICATION

In Application mode, you can use the keypad keys in key definition instructions. For a list of the keys that you can define in Application mode, see the DEFINE KEY AS and PROGRAM KEY IS instructions.

Notes

Once a keypad mode is set by a call to a request, it remains in that mode for the life of the TDMS application or until another KEYPAD MODE IS instruction is executed.

KEYPAD [MODE] IS

When you specify a KEYPAD key name with the DEFINE KEY AS or PROGRAM KEY IS instruction, the key name can be enabled only if you place the terminal keypad in Application mode with a KEYPAD IS APPLICATION instruction.

When the keypad mode is Numeric, the keypad keys are not recognized as the key functions specified in the key definition instructions. When a keypad key is pressed under Numeric mode, TDMS translates the key stroke as numeric data; no warning message is issued.

Examples

```
RDUDFN> KEYPAD IS NUMERIC;
```

Places the keypad in Numeric mode for the duration of the application or until TDMS executes another KEYPAD IS instruction.

```
RDUDFN> KEYPAD MODE IS APPLICATION;
```

Places the keypad in Application mode for the duration of the application or until TDMS executes another KEYPAD IS instruction.

If you embed single or double quotation marks within a quoted string, obey the following rules:

- If the string is enclosed within single quotation marks, use either:
  - Double quotation marks within the string:

    'system "down" at 5:00 p.m.'
  - Two sets of single quotation marks within the string:

    'system ''down'' at 5:00 p.m.'
- If the string is enclosed within double quotation marks, use either:
  - Single quotation marks within the string:

    "system 'down' at 5:00 p.m."
  - Two sets of double quotation marks within the string:

    "system ""down"" at 5:00 p.m."

Notes

You can use this instruction within the body of a request to display error information or inform the operator of events (system shutdown and so on).

TDMS displays the data on line 24 (or line 14 if the terminal is currently set in 132-column mode and has no Advanced Video Option).

See the PROGRAM KEY IS instruction for how to use the MESSAGE LINE IS instruction within a PRK.

Examples

```
RDUDFN> MESSAGE LINE IS "System shutdown at 5 p.m.";
```

A system message is displayed on the message line of the terminal.

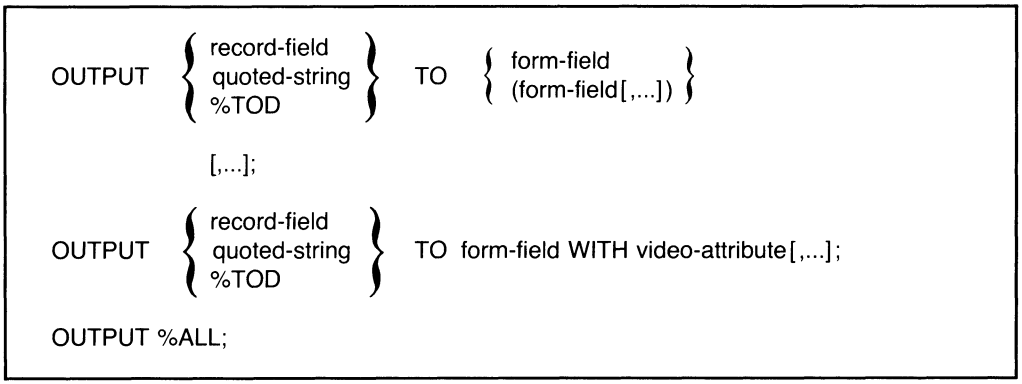
```
RDUDFN> MESSAGE LINE IS "Employee number does not exist";
```

An application-specific message is displayed on the message line of the terminal.

3.16 OUTPUT TO Instruction

Displays the specified data in one or more form fields.

Format



Prompt

```
RDUDFN>
```

Instruction Parameters

record-field

The name of a record field from which TDMS copies data. You must specify preceding group field names only if they are necessary to make the reference unique.

In RDU, the record name is treated as the top-level group field name. Any record name you use must be specified in the RECORD IS instruction. If a unique name is specified in the WITH NAME modifier of the RECORD IS instruction, you must use the unique name.

RDU always searches all the records you specify in the RECORD IS instruction for a record field, whether or not you specify the record name.

Even when you use a record name, you cannot always access a record field name. For more information, see Chapter 6, Rules for Resolving Ambiguous Field References.

quoted-string

Any string of characters enclosed in either single or double quotation marks. The length of the string cannot be greater than the size of the receiving form field or extend beyond a single line. Therefore, it cannot be over 80 characters long. You must use matching punctuation at the beginning and end of the string ("text" or 'text' but not "text' or 'text").

If you embed single or double quotation marks within a quoted string, obey the following rules:

- If the string is enclosed within single quotation marks, use either:
  - Double quotation marks within the string:

    'system "down" at 5:00 p.m.'

  - Two sets of single quotation marks within the string:

    'system ''down'' at 5:00 p.m.'

- If the string is enclosed within double quotation marks, use either:
  - Single quotation marks within the string:

    "system 'down' at 5:00 p.m."

  - Two sets of double quotation marks within the string:

    "system ""down"" at 5:00 p.m."

form-field

The name of a field on the active form. If you specify a list of form fields, you must enclose them in matching parentheses and separate the field names with commas.

%TOD

The current time or date. The format of the value TDMS returns depends on the format you give the field. If you map %TOD to a time field, TDMS gives a value in time format. If you map %TOD to a date field, TDMS gives a value in date format.

%ALL

All the record fields that have identically named form fields on the active form.

Instruction Modifier

WITH video-attribute

The keyword WITH and one or more video attributes that you can specify for the form field, including:

- [NO] BLINK
- [NO] BOLD
- [NO] REVERSE
- [NO] UNDERLINE

You can specify video attributes only when you use the simplest format of the OUTPUT TO instruction. For example:

```
RDUDFN> OUTPUT EMPLOYEE TO EMPLOYEE WITH UNDERLINE;
```

If you specify more than one video attribute, they must be separated by commas.

The video modifier is ignored if you run a TDMS application on a VT52 terminal.

Notes

OUTPUT TO is one of three request instructions that move data between a form and record; the others are INPUT TO and RETURN TO.

TDMS executes all OUTPUT TO instructions before it executes any other mapping instructions.

In an explicit mapping, RDU checks (in the default Validate mode) the record and form fields you specify to see that:

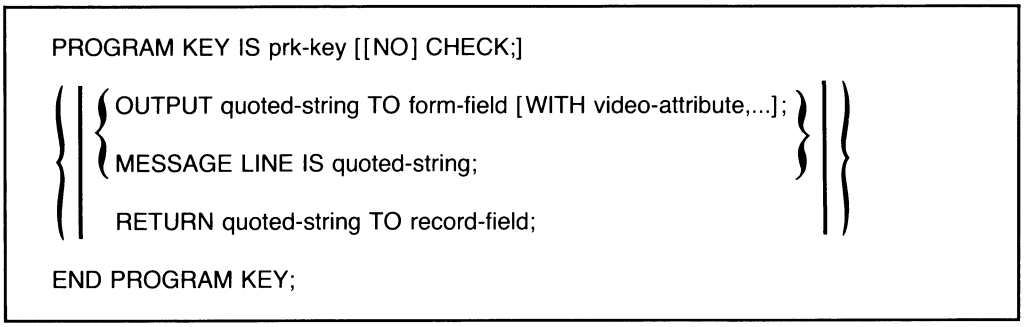
- All the fields exist in the record and form definitions in the CDD
- The fields do not exist more than once in the records or form used by a request
- The mappings defined are valid (field data types, structures, lengths, sign conditions, and so on, are compatible)

If RDU finds errors (in the default Validate mode), it returns an error level message and does not create (or replace, modify, or validate) a request.

3.17 PROGRAM KEY IS Instruction

Specifies a program request key (PRK) and the resulting instructions for TDMS to execute when the operator presses the PRK.

Format



Prompt

```
RDUDFN>
```

Instruction Parameters

prk-key

The name of the program request key that you specify. There are several categories of key names.

- The KEYPAD keys are those on the numeric keypad at the right edge of the keyboard. There is a numeric keypad on VT100- and VT200-series terminals. Remember that the KEYPAD keyword is *not* enclosed in quotation marks but the remaining part of the prk-key parameter is.

  Note that the keypad must be set to Application mode when KEYPAD keys are used in a request. You use the KEYPAD MODE IS instruction to set the keypad to Application mode.

The following numeric keypad key names can be specified with the KEYPAD keyword:

0	4	8
1	5	9
2	6	. (period)
3	7	, (comma)
		- (hyphen)

- PF keys are located on the numeric keypad on VT100- and VT200-series terminals. You do not include the KEYPAD keyword as part of the prk-key parameter. The PF keys that you can specify are:

  PF1
  PF2
  PF3
  PF4

- The F (function) keys are located across the top row of VT200-series keyboards. Keys F1 through F5 are local function keys that cannot be redefined. You can specify the other F keys with the PROGRAM KEY IS instruction. When specifying an F key name, do not separate the F from the digit. You can specify the following F keys:

F6	F10	F14	F18
F7	F11	F15	F19
F8	F12	F16	F20
F9	F13	F17	

- Only VT200-series terminals have E keys. These six keys are located on the "editing" keypad, above the arrow keys. The E keys that you can specify are:

E1	E4
E2	E5
E3	E6

- You can use the keywords listed below to specify the arrow keys. In addition, you can specify the GOLD keyword with arrow key names. (Note that when you use GOLD in combination with an arrow keyword, you do not enclose the keyword in quotation marks.)

DOWNARROW	GOLD DOWNARROW
LEFTARROW	GOLD LEFTARROW
RIGHTARROW	GOLD RIGHTARROW
UPARROW	GOLD UPARROW

- There are other key names that you can use for the prk-key parameter. The keys are:

  BACKSPACE (VT100 mode)
  ENTER
  LINEFEED (VT100 mode)
  RETURN
  TAB

  Note that you should specify BACKSPACE and LINEFEED only for terminals in VT100 mode. When using VT200-mode, you specify the F12 and F13 keys instead of BACKSPACE and LINEFEED as the prk-key.

  If you plan to redefine the ENTER key, be sure to set the keypad to Application mode. When the keypad is in Numeric mode, the ENTER key has the same definition as the RETURN key. When the keypad is in Application mode, you can define the ENTER key to have a different function from the RETURN key.

- The keyword GOLD can be used with many key names. The GOLD keyword is *not* enclosed in quotation marks in the prk-key parameter, but the key name is, for example, GOLD "&" (GOLD ampersand) and GOLD " " (GOLD space). The default GOLD key is the PF1 key on the numeric keypad.

  The GOLD keyword can be used with alphanumeric keys as well as many character keys. Note that uppercase and lowercase letters are interpreted as the same key.

  The alphanumeric keys include:

  A-Z
  a-z
  0-9

You can specify the following characters with GOLD:

&	(ampersand)	%	(percent)
*	(asterisk)	.	(period)
@	(at sign)	+	(plus sign)
\	(backslash)	?	(question mark)
^	(circumflex)	"	(quotation mark)
:	(colon)	>	(right angle bracket)
,	(comma)	}	(right brace)
$	(dollar sign)	)	(right parenthesis)
=	(equal sign)	]	(right square bracket)
!	(exclamation point)	;	(semicolon)
`	(grave accent)	'	(single quotation mark)
-	(hyphen)	/	(slash)
<	(left angle bracket)		(space)
{	(left brace)	~	(tilde)
(	(left parenthesis)	_	(underscore)
[	(left square bracket)	|	(vertical line)
#	(number sign)		

OUTPUT quoted-string TO form-field

Specifies a text string to be written to a form field when the operator presses the named PRK. You cannot specify both an OUTPUT quoted-string and a MESSAGE LINE IS quoted-string in the same PROGRAM KEY IS instruction.

WITH video-attribute

The keyword WITH and one or more video attributes you can specify for the form field, including:

[NO] BOLD
[NO] REVERSE
[NO] UNDERLINE
[NO] BLINK

Use a comma to separate each video attribute in a list.

MESSAGE LINE IS quoted-string

A text string to be written to the message line of the terminal when the operator presses the named PRK. You cannot specify both a MESSAGE LINE IS quoted-string and an OUTPUT quoted-string. If you do, RDU will signal an error.

RETURN quoted-string TO record-field

A text string written to a program record field when an operator enters the program request key named in the PROGRAM KEY IS instruction.

quoted-string

Any string of characters enclosed in either single or double quotation marks. The string cannot be larger than the field to which it is mapped. In addition, it cannot extend beyond a single line within your request. Therefore, it cannot be over 80 characters long. You must use matching punctuation at the beginning and end of the string ("text" or 'text' but not 'text" or "text').

If you embed single or double quotation marks within a quoted string, obey the following rules:

- If the string is enclosed within single quotation marks, use either:
  - Double quotation marks within the string:

    'system "down" at 5:00 p.m.'

  - Two sets of single quotation marks within the string:

    'system ''down'' at 5:00 p.m.'

- If the string is enclosed within double quotation marks, use either:
  - Single quotation marks within the string:

    "system 'down' at 5:00 p.m."

  - Two sets of double quotation marks within the string:

    "system ""down"" at 5:00 p.m."

form-field

The name of one form field in which the text string is displayed when the operator presses the PRK. Only one form field or form field element in an array can be specified. The form field must be large enough to contain the text string it is to receive.

record-field

The name of one field in a program record. Only one record field or record field element in an array can be specified. It must be large enough to contain the text string it is to receive.

Instruction Modifiers

CHECK

You can specify the modifier CHECK or NO CHECK to the PROGRAM KEY IS instruction. The default is CHECK.

When the operator enters a PRK, TDMS checks to see that all fields defined as Response Required in the form definition (that are also mapped for input) do indeed have data entered in them. If a Response Required field does not have data in it, TDMS ignores the PRK.

If the PRK is pressed while the cursor is in a field, TDMS checks that, if the field is a Must Fill field, the operator has filled the field. TDMS also checks any field validators associated with the field.

When TDMS terminates the request, it returns data from all the form fields that are mapped for input and return to the record. That data may be either:

- Data entered in the form fields during the current call to the request
- Data mapped to the form fields by the current or previous call to the request
- Data associated with the form fields by form definition defaults (if no other data is in the fields)

NO CHECK

Allows you to specify that TDMS terminate a request call without checking if Response Required fields have data in them.

If you assign the NO CHECK modifier, TDMS executes only the instructions within the PROGRAM KEY IS instruction and terminates the request. The only data TDMS returns, therefore, is the data specified in a RETURN TO instruction within the PROGRAM KEY IS instruction. It does not return any data from other INPUT TO or RETURN TO instructions in the request.

Notes

You can use a PRK to:

- Output a text string to a form field on an active form
- Modify the video attributes of the form field to which you output data on an active form
- Return a fixed string to a field in the program record
- Write a fixed string to the message line of a terminal

The OUTPUT TO, MESSAGE LINE IS, and RETURN TO instructions within a PROGRAM KEY IS instruction are referred to as PRK instructions.

You cannot specify both an OUTPUT TO instruction and a MESSAGE LINE IS instruction in a single PROGRAM KEY IS instruction. When an application program runs and when the operator enters a key defined as a program request key, TDMS executes the program request key instructions and terminates the request call.

You do not specify a semicolon at the end of the PROGRAM KEY IS prk-key instruction line except when you use the [NO] CHECK modifier.

RDU cannot determine the keypad mode at run time, so RDU does not check the mode when it validates the request. (See the KEYPAD MODE IS instruction.)

When keys or key sequences have more than one definition, you need to be aware that TDMS processes only one of the key definitions. These are the precedence rules that TDMS uses to resolve multiple definitions:

- If a key or key sequence is defined as an application function key (AFK), only the AFK function is ever executed. TDMS ignores any key definition instructions (PROGRAM KEY IS or DEFINE KEY AS) for that key or key sequence.
- If there is no AFK definition for a key or key sequence, a key definition instruction within a conditional instruction takes precedence and is executed. A key definition instruction in the base request is ignored.
- If no key definition instruction occurs within a conditional instruction, a key definition instruction in the base request is executed.

- If two or more key definition instructions occur in the base request or in conditional instructions defined at the same level within a request, one of the instructions takes precedence and is executed. However, there are no rules to determine which key definition instruction prevails in this instance.

For example, if you define BACKSPACE as a PRK using the PROGRAM KEY IS instruction and also have an AFK for CTRL/H, the AFK definition is the one that is processed when the operator presses CTRL/H or BACKSPACE.

Examples

```
RDUDFN>  PROGRAM KEY IS GOLD "C" NO CHECK;
RDUDFN>     MESSAGE LINE IS "Cancel Operation";
RDUDFN>  END PROGRAM KEY;
```

When the operator presses the key sequence GOLD-C, TDMS displays the text string Cancel Operation on line 24 of the screen and terminates the call to the request.

```
RDUDFN>  KEYPAD MODE IS APPLICATION;
RDUDFN>  PROGRAM KEY IS KEYPAD "9"
RDUDFN>     OUTPUT "Canceling Update" TO MESSAGE_FIELD WITH BOLD,BLINK;
RDUDFN>     RETURN "Cancel" TO RECORD1.ACTION;
RDUDFN>  END PROGRAM KEY;
```

When the operator presses the keypad key 9 at run time, TDMS checks that all Response Required fields on the active form have data entered in them. If they do, TDMS outputs a message to the form field MESSAGE_FIELD and bolds and blinks that field. It also returns the message Cancel to the record field ACTION in the record RECORD1 and then terminates the request.

Note that you can specify a list of video field attributes in the OUTPUT TO instruction within a PROGRAM KEY IS instruction.

```
RDUDFN> PROGRAM KEY IS F20
RDUDFN>       NO CHECK;
RDUDFN>       MESSAGE LINE IS "Operator pressed key F20.";
RDUDFN>       RETURN "F20" TO ACTION;
RDUDFN> END PROGRAM KEY;
```

When the operator presses the F20 key, TDMS displays the text string "Operator pressed key F20" on line 24 of the screen. The text string F20 is returned to the record field ACTION. Then TDMS terminates the request.

3.18 RECORD IS Instruction

Identifies the CDD record or records to and from which you map data.

Format

```
{ RECORD IS   }
{ RECORDS ARE }  record-path-name  [WITH NAME unique-record-name],...;
```

Prompt

```
RDUDFN>
```

Instruction Parameter

record-path-name

The CDD path name (given, relative, or full) of an existing record definition. Note that the name you use in a request is usually the same as the path name stored in the CDD directory. You can, however, use a logical name different from the CDD path name.

Two records, or a record and a form, cannot use the same name within the body of a request. If two record given names, or a record given name and a form given name, are the same, you must specify a unique name using the WITH NAME modifier. Otherwise, RDU displays an error message and does not process the request.

Instruction Modifier

WITH NAME unique-record-name

The keywords WITH NAME and a name which no other record or form can have within the request. You must use the WITH NAME clause to specify a unique record name if two records, or a record and a form, in your request have the same given name. The unique record name must conform to the rules for a CDD given name.

The unique record name, if specified, is the one you must use in subsequent mapping instructions within the body of a request. If the unique name is not specified, you use the given name.

Example

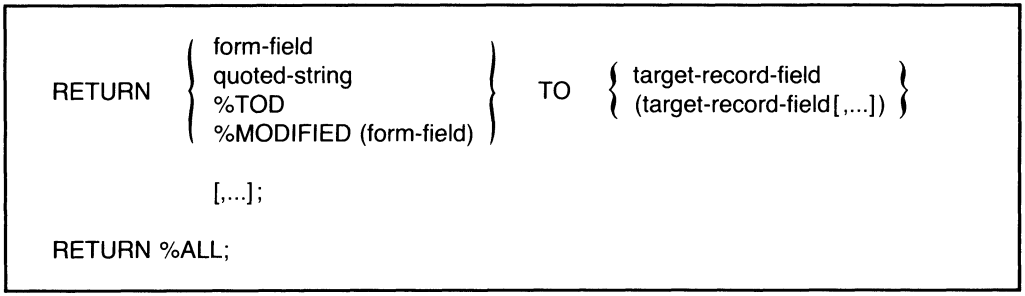
```
RDUDFN> RESET FIELD NAME, BADGE, SEX;
```

Resets form fields NAME, BADGE, and SEX to their form-defined video defaults.

3.21 RETURN TO Instruction

Returns data to one or more record fields. TDMS does not place the cursor in the form field named or allow the operator to enter data in that field.

Format



Prompt

```
RDUDFN>
```

Instruction Parameters

form-field

The name of the field on the active form.

quoted-string

The string is identified within the body of a request and is returned to the record field. It cannot be larger than the size of the record fields to which it is returned. A quoted string cannot extend beyond a single line. Therefore, it cannot be over 80 characters long. You must use matching punctuation at the beginning and end of the string ("text" or 'text' but not 'text" or "text').

If you embed single or double quotations marks within a quoted string, obey the following rules:

- If the string is enclosed within single quotation marks, use either:
  - Double quotation marks within the string:

    'system "down" at 5:00 p.m.'

  - Two sets of single quotation marks within the string:

    'system ''down'' at 5:00 p.m.'

- If the string is enclosed within double quotation marks, use either:
  - Single quotation marks within the string:

    "system 'down' at 5:00 p.m."

  - Two sets of double quotation marks within the string:

    "system ""down"" at 5:00 p.m."

target-record-field

The name of one or more record fields to which TDMS returns data. If you specify a list of record fields, you must enclose the list in matching parentheses and separate the fields with commas.

In RDU, the record name is treated as the top-level group field name. Any record name you use must be specified in the RECORD IS instruction. If a unique name is specified in the WITH NAME modifier of the RECORD IS instruction, you must use the unique name.

RDU always searches all the records you specify in the RECORD IS instruction for a record field, whether or not you specify the record name.

Even when you use a record name, you cannot always access a record field name. For more information, see Chapter 6, Rules for Resolving Ambiguous Field References.

%TOD

The current system time in 64-bit format. TDMS returns the system time to the specified record field. The record field must have a data type of ADT.

%MODIFIED (form-field)

A value indicating whether an operator has modified a field. When the specified form field has been modified, TDMS returns a 1 to the target record field. When the specified form field has not been modified, TDMS returns a 0 to the target record field.

TDMS uses the following criteria to determine whether a form field is modified or not during a request call:

- At the start of a request instance, no fields are considered to be modified.
- Neither an OUTPUT TO or DEFAULT FIELD instruction causes a form field to be set to modified.
- As soon as an operator enters a character into a form field that changes the original form field contents, the modified flag is set. The flag remains set even if the operator then restores the original contents of the field. Note that changing a form field's contents from uppercase to lowercase constitutes modifying the field.
- If an operator presses the LINE FEED key while in a form field, TDMS considers the field to be modified.

%ALL

All the fields on the active form that have identically named record fields.

Notes

RETURN TO is one of three request instructions that move data between a form and record; the others are OUTPUT TO and INPUT TO.

Unlike the INPUT TO instruction, the RETURN TO instruction does not open the field for input by the operator. If you specify a form field, the RETURN TO instruction returns one of the following:

- The data output to the field in the current call to a request
- The form field contents from the immediately previous request call
- The form field default assigned in the form definition (if no other data is in the field)

If a form field is mapped by both a RETURN TO and INPUT TO instruction in the same request, the data returned to the program is the result of the INPUT TO instruction.

TDMS executes all other instructions in the request before it executes the RETURN TO instruction.

In an explicit mapping, RDU checks (in the default Validate mode) that:

- The form and record fields you specify exist in the record and form definitions in the CDD
- The fields do not exist more than once in the records used by a request
- The mappings defined are valid (field data types, structures, lengths, sign conditions, scale factors, and so on are compatible)

If RDU finds errors (in the default Validate mode), it returns an error level message and does not create (or replace, modify, or validate) a request.

You can use the RETURN %ALL instruction if you want to return data from all the fields on an active form to identically named record fields.

In a %ALL mapping, RDU does not create the *individual mapping* if:

- A form field does not have an identically named record field in the records used by a request
- An identically named record field exists more than once in the records used by a request
- The mappings defined are not valid (field data types, structures, lengths, sign conditions are not compatible)

RDU does, however, create the request, unless *all* the mappings implied by the %ALL syntax are incorrect.

If /LOG is specified, %ALL mappings will appear:

- In the listing file (if any)
- In the output file or device defined as SYS$OUTPUT

%ALL mappings will appear in the log file if the SET LOG command and the /LOG qualifier are specified.

Examples

```
RDUDFN>  RETURN "Gone" TO Wk_MSG_RECORD.MESSAGE_FIELD;
```

Copies the contents of the quoted string to the record field MESSAGE_FIELD in the record or group field WK_MSG_RECORD after completing all other instructions in the request.

```
RDUDFN>  RETURN NAME  TO EMP_NAME,
RDUDFN>         BADGE TO (EMP_BADGE, EMP_NUMBER);
```

Returns the contents of the form field NAME to the record field EMP_NAME and the contents of the form field BADGE to two record fields, EMP_BADGE and EMP_NUMBER. The contents will be either the form-defined default or the data collected from the immediately previous request call.

```
RDUDFN>  FORM IS PERSONNEL_FORM;
RDUDFN>  RECORD IS PERSONNEL_RECORD;
RDUDFN>      DISPLAY FORM PERSONNEL_FORM;
RDUDFN>      RETURN %ALL;
```

Returns the data in all the form fields on the PERSONNEL_FORM that have identically named record fields in the record PERSONNEL_RECORD to those record fields.

3.22 [NO] REVERSE FIELD Instruction

Sets or clears the reverse video attribute of a field on an active form.

Format

```
[NO] REVERSE FIELD { form-field[,...] } ;
                   { %ALL             }
```

Prompt

```
RDUDFN>
```

Instruction Parameters

form-field

The name assigned to the form field. The field must be on the active form. You can specify a single field or a list of form fields separated by commas.

%ALL

All the fields on the active form.

Notes

Reverse affects the video screen background of a form field and changes it to the opposite of the previous setting. If the field is dark, TDMS reverses it to light. If it is light, TDMS reverses it to dark.

If you specify the REVERSE FIELD or NO REVERSE FIELD instruction in a request, it overrides:

- A Reverse or No Reverse attribute assigned in a form definition.
- A REVERSE FIELD or NO REVERSE FIELD instruction that is still active from a previous request call. A video instruction is still active when:
  - A form is still on the screen from a previous request call
  - The current call to a request uses that same form with a USE FORM instruction

[NO] REVERSE FIELD

At run time, a REVERSE or NO REVERSE instruction used within a conditional instruction supersedes one in a base request or any outer conditional instruction.

If a field has been defined with input highlighting in FDU, the [NO] REVERSE instruction will override the input highlighting video attribute for that field. Any other video attributes assigned to the field will be unaffected by the [NO] REVERSE instruction.

The REVERSE instruction is ignored if you run a TDMS application on a VT52 terminal.

Example

```
RDUDFN> REVERSE FIELD EMPLOYEE_NAME;
```

Reverses the screen background of the form field EMPLOYEE_NAME.

3.26 [NO] UNDERLINE FIELD Instruction

Sets or clears the underline video attribute of a field on an active form.

Format

```
[NO] UNDERLINE FIELD { form-field[,...] } ;
                     { %ALL             }
```

Prompt

```
RDUDFN>
```

Instruction Parameters

form-field

The name assigned to the form field. The field must be on the active form. You can specify one form field or a list of form fields separated by commas.

%ALL

All the fields on the active form.

Notes

If you specify the UNDERLINE FIELD or NO UNDERLINE FIELD instruction in a request, it overrides:

- An Underline or No Underline attribute assigned in a form definition.
- An UNDERLINE FIELD or NO UNDERLINE FIELD instruction that is still active from a previous request call. A video instruction is still active when:
  - A form is still on the screen from a previous request call
  - The current call to a request uses that same form with a USE FORM instruction

At run time, an UNDERLINE or NO UNDERLINE instruction used within a conditional instruction supersedes one in a base request or any outer conditional instruction.

[NO] UNDERLINE FIELD

If a field has been defined with input highlighting in FDU, the [NO] UNDERLINE instruction will override the input highlighting video attribute for that field. Any other video attributes assigned to the field will be unaffected by the [NO] UNDERLINE instruction.

The UNDERLINE instruction is ignored if you run a TDMS application on a VT52 terminal.

Examples

```
RDUDFN>  UNDERLINE FIELD  NAME, SEX, BADGE;
```

Underlines the form fields NAME, SEX, and BADGE.

```
RDUDFN>  UNDERLINE FIELD %ALL;
```

Underlines all the form fields on the active form.

3.28 [NO] WAIT Instruction

Displays a form until the operator presses any request termination key.

Format

```
[NO] WAIT;
```

Prompt

```
RDUDFN>
```

Notes

TDMS does not complete the request and return to the program until the operator presses a request termination key. The request termination key can be any key defined to perform the following functions:

Key functions	Default keys
DONE	RETURN, ENTER, PRKs
NEXT	TAB
PREVIOUS	BACKSPACE
EXIT_SCROLL_DOWN	GOLD DOWNARROW
EXIT_SCROLL_UP	GOLD UPARROW
SCROLL_DOWN	DOWNARROW
SCROLL_UP	UPARROW

Use the WAIT instruction if the request contains no input mappings. If you do not use the WAIT instruction, TDMS may display the output mappings so quickly that the operator does not see the data displayed on the form.

The WAIT instruction is not necessary if a request contains an INPUT instruction. If you use a WAIT instruction in a request containing this instruction, TDMS ignores the WAIT instruction.

Example

```
RDUDFN> DISPLAY FORM EMPLOYEE;
RDUDFN> OUTPUT %ALL;
RDUDFN> WAIT;
```

TDMS displays all the data to the fields on the EMPLOYEE form and waits until the operator presses a request termination key.

key-astadr

The address of a routine in the application program; you pass this parameter by reference. When the operator presses a declared AFK, TDMS will call this routine at AST level. The user routine must have the following calling sequence:

```
status.wlc.v = ROUTADR (key-astprm.rlu.v
                        ,channel.rlu.r
                        ,key-id.rlu.r)
```

You can use the AST service routine with or without an AST parameter. You can also use the AST service routine with an event flag. Note that the key-astprm parameter is passed to the AST routine by value.

key-astprm

The longword that contains the AST parameter to be passed to the AFK service routine; you pass this optional parameter by reference. If the AST parameter is not present, and a service routine is, TDMS will pass an AST parameter of 0 to the service routine.

You can pass any type of parameter you would like your AST routine to receive, including addresses.

Return Status

Ret-status is the standard VAX/VMS return status indicating the success or failure of the call. The codes that can be returned on this call are:

TSS$_BUGCHECK

Fatal internal software error (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVKEYID

Invalid key id (F)

TSS$_NORMAL

Normal successful completion (S)

TSS$_SYNASTLVL

Synchronous calls may not be called at AST level (F)

Notes

Application function keys (AFKs) provide exception notification services for terminal-related events. During execution of a TDMS application, the operator can press AFK keys in order to initiate actions outside the context of the current input to the active form.

AFKs are asynchronous function keys; that is, they operate independently of requests. As asynchronous function keys, AFKs initiate asynchronous processing in the user's application program.

You can enable and disable the operator's use of AFKs by issuing TSS$DECL_AFK and TSS$UNDECL_AFK calls in the application program. The TSS$DECL_AFK call specifies the AFK by the key-id parameter and associates that key with a service routine, an event flag, or both. After the program has made a TSS$DECL_AFK call, the operator can press the enabled AFK whenever he wishes to invoke a special function, until the key is disabled:

- When the application program issues a matching TSS$UNDECL_AFK or TSS$UNDECL_AFK_A call
- When the application program closes the channel with a TSS$CLOSE or TSS$CLOSE_A call
- Automatically, when the application program ends

TSS$_NORMAL

Normal successful completion (S)

TSS$_SYNASTLVL

Synchronous calls may not be called at AST level (F)

Notes

The reserved message line is usually the last line on the screen.

If you are displaying a 132-column form on a terminal without the AVO option, the reserved message line is line 14.

Messages are limited to 80 characters unless a form with 132 columns is currently displayed. The message can then have up to 132 characters. The message remains on the screen until the operator presses a terminator key, such as one that signals the completion of input to a field or completion of a wait.

Note that you can mix synchronous and asynchronous calls. For example, you can use TSS$OPEN_A with TSS$CLOSE or TSS$DECL_AFK_A with TSS$UNDECL_AFK.

Examples

BASIC

```
Return_status = TSS$READ_MSG_LINE(Channel, &
                                  Response_text, &
                                  Message_prompt, &
                                  Response_length)
```

COBOL

```
CALL "TSS$READ_MSG_LINE"
     USING BY REFERENCE Channel,
           BY DESCRIPTOR Response-text,
           BY DESCRIPTOR Message-prompt,
           BY REFERENCE Response-length,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$READ_MSG_LINE(%REF(Channel),
1                                  %DESCR(Response_text),
2                                  %DESCR(Message_prompt),
3                                  %REF(Response_length))
```

Table 5-3: TDMS Application Function Keys (AFKS)

Key Id	Control Key	Key Id	Control Key
0	CTRL/space bar	15	CTRL/O
1	CTRL/A	16	CTRL/P
2	CTRL/B	18	CTRL/R
3	CTRL/C	20	CTRL/T
4	CTRL/D	21	CTRL/U
5	CTRL/E	22	CTRL/V
6	CTRL/F	23	CTRL/W
7	CTRL/G	24	CTRL/X
8	CTRL/H	25	CTRL/Y
9	CTRL/I	26	CTRL/Z
10	CTRL/J	27	CTRL/[
11	CTRL/K	28	CTRL/backslash
12	CTRL/L	29	CTRL/]
13	CTRL/M	30	CTRL/~
14	CTRL/N	31	CTRL/?

key-efn

The address of a longword containing the number of the event flag that is to be set when the AFK is pressed; you pass this parameter by reference. If the parameter is not present, TDMS does not set an event flag when the operator presses the key.

You may use the event flag by itself or together with an AST service routine.

key-astadr

The routine in the application program; you pass this parameter by reference. When the operator presses a declared AFK, VAX TDMS calls this routine at AST level. The user routine must have the following calling sequence:

```
status.wlc.v = ROUTADR (key-astprm.rlu.v
                        ,channel.rlu.r
                        ,key-id.rlu.r)
```

You can use the AST service routine with or without an AST parameter. You can also use the AST service routine with an event flag. Note that the key-astprm parameter is passed to the AST routine by value.

key-astprm

The longword that contains the AST parameter to be passed to the AFK service routine; you pass this optional parameter by reference. If the AST parameter is not present, and a service routine is, TDMS will pass an AST parameter of 0 to the service routine.

You can pass any type of parameter you would like your AST routine to receive, including addresses.

Return Status and/or Completion Code (RSB)

Ret-status is the standard VAX/VMS return status indicating the success or failure of the call.

The return status for an asynchronous call, if successful, indicates only that the call was initiated, not that it was completed.

The codes that can be returned on this call are:

TSS$_BUGCHECK

Fatal internal software error (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

Notes

The reserved message line is usually the last line on the screen. If you are displaying a 132-column form on a terminal without the AVO option, the reserved message line is line 14. Messages are limited to 80 characters unless a form with 132 columns is currently displayed. The message can then be 132 characters. When the operator presses the RETURN key or any other request processing key, the message line is cleared.

Messages are limited to 80 characters unless a form with 132 columns is currently displayed. The message can then have up to 132 characters. The message remains on the screen until the operator presses a terminator key, such as one that signals the completion of input to a field or completion of a wait.

An asynchronous call initiates a TDMS operation and then returns control immediately to the application program. When the operation is finished, TDMS notifies the application program by:

- Declaring the user's asynchronous system trap (AST) routine
- Setting an event flag specified by the user
- Both declaring the user's AST routine and setting the event flag specified by the user

Asynchronous calls can be made from AST level as well as non-AST level.

Except for TSS$CANCEL, synchronous calls cannot be made from AST level. Making a synchronous call to TDMS from an AST routine will cause an error to be returned.

Note that you can mix synchronous and asynchronous calls. For example, you can use TSS$OPEN_A with TSS$CLOSE, or TSS$DECL_AFK_A with TSS$UNDECL_AFK.

Examples

BASIC

```
EXTERNAL LONG Ast_routine
            .
            .
            .

Reutrn_status = TSS$READ_MSG_LINE_A(%REF(Channel) &
                                    Return_status_block, &
                                    Event_flag_number, &
                                    Ast_routine, &
                                    Ast_parameter BY VALUE, &
                                    Response_text, &
                                    Message_prompt, &
                                    Response_length)
```

COBOL

```
CALL "TSS$READ_MSG_LINE_A"
     USING BY REFERENCE Channel,
           BY REFERENCE Return-status-block,
           BY REFERENCE Event-flag-number,
           BY REFERENCE Ast-routine,
           BY VALUE Ast-parameter,
           BY DESCRIPTOR Response-text,
           BY DESCRIPTOR Message-prompt,
           BY REFERENCE Response-length,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$READ_MSG_LINE_A(%REF(Channel),
1                                    %REF(Return_status_block),
2                                    %REF(Event_flag_number),
3                                    %REF(Ast_routine),
4                                    Ast_parameter,
5                                    %DESCR(Response_text),
6                                    %DESCR(Message_prompt),
7                                    %REF(Response_length))
```

Instruction Execution Order 7

For the most part, request instructions can appear in any order. However, there are a few rules to follow:

- Form header instructions must come before other instructions.
- The END DEFINITION instruction must be the last instruction.

Regardless of the order in which you specify the request instructions, TDMS executes instructions in the following order:

- First, TSS$REQUEST evaluates all CONTROL FIELD IS instructions. It attempts to match all control values with the case values specified under the CONTROL FIELD IS instruction. It then gathers all request instructions that are to be executed during a request call.
- Next, TDMS evaluates, but does not execute, any DEFINE KEY AS and PROGRAM KEY IS instructions.
- Then TDMS evaluates and executes all OUTPUT operations:
  - Request-wide operations (CLEAR SCREEN, SIGNAL MODE IS, KEYPAD MODE IS, and so on)
  - Form field setup operations (output mappings, DEFAULT FIELD, RESET FIELD, video change operations such as Bold field)
  - DISPLAY FORM or USE FORM

- Next, TDMS evaluates and executes all INPUT operations:
  - All form fields mapped for input are opened for operator input.

    Note that any program request key can be pressed during any input operation. The request instructions associated with any program request keys (except RETURN) will be executed when the program request key is pressed.
  - WAIT instruction.
- Finally, TDMS evaluates and executes all RETURN operations.

ILLDSTLEN **destination length must be greater than 8**

Severity: Explicit: Error (E)
%ALL: Information (I)

Explanation: A mapping error. A data type conversion error occurred because the length of a receiving field is less than 8 (too short for the mapping to be valid).

User Action: Make the length of the receiving field greater than 8.

ILLFLDDAT **illegal field datatype**

Severity: Explicit: Error (E)
%ALL: Information (I)

Explanation: A mapping message. The data type of the field is not supported by TDMS.

User Action: Define the field to have a valid TDMS data type.

ILLKBDKEY **program key <key-name> is not a legal keyboard key**

Severity: Error (E)

Explanation: A syntax message. A PROGRAM KEY IS instruction specifies a program request key that is not one of the valid keys.

User Action: Specify one of the valid keys as the program request key in the PROGRAM KEY IS instruction.

ILLKEYFNC **illegal key function for DEFINE KEY AS**

Severity: Error (E)

Explanation: A syntax message. You specified a key function that is not valid in a DEFINE KEY AS instruction.

User Action: Specify a key function that is valid in a DEFINE KEY AS instruction.

ILLKPDKEY **program key <key-name> is not a legal keypad key**

Severity: Error (E)

Explanation: A syntax message. A PROGRAM KEY IS KEYPAD instruction specifies as the program request key a keypad key that is not one of the valid keys on the keypad.

User Action: Specify one of the digits 0 - 9, comma, period, or hyphen as the program request key in the PROGRAM KEY IS KEYPAD instruction.

ILLLEDNO **LED number must be between 1 and 4**

Severity: Error (E)

Explanation: A syntax message. You specified a LED number in a LIGHT LIST instruction that is less than 1 or greater than 4.

User Action: Specify a LED number that is between 1 and 4 in the LIGHT LIST instruction.

ILLLITNUM **light number <number> is invalid**

Severity: Error (E)

Explanation: A mapping message. You specified a LED number in a LIGHT LIST instruction that is less than 1 or greater than 4.

User Action: Specify a LED number that is between 1 and 4 in the LIGHT LIST instruction.

ILLMSGLIN **multiple message lines declared**

Severity: Error (E)

Explanation: A syntax message. More than one MESSAGE LINE IS instruction appears in the base part of a request or within a single case value in a CONTROL FIELD IS instruction.

User Action: Specify only one MESSAGE LINE IS instruction.

ILLNAME **Form name or field name <text> is not valid CDD name**

Severity: Error (E)

Explanation: A mapping message. The name of the form or form field is not in the CDD.

User Action: Check that the form definition is in the CDD and/or contains the field name you use in the mapping reference.

ILLOFFSET **display offset number must be between 0 and +22**

Severity: Error (E)

Explanation: A syntax message. An offset in a USE FORM WITH OFFSET or DISPLAY FORM WITH OFFSET instruction is less than 0 or greater than +22.

User Action: Specify an offset number in the USE FORM or DISPLAY FORM instruction that is between 0 and +22.

ILLPASSCHR **illegal character in CDD password**

Severity: Error (E)

Explanation: A syntax message. An illegal character or escape sequence is in a password associated with a path name in a FORM IS, RECORD IS, or REQUEST IS instruction.

User Action: Remove the illegal character from the password.

ILLPERCENT **illegal percent character in text**

Severity: Error (E)

Explanation: A syntax message. The first character in an item is a percent sign, and the item is not a %INCLUDE, %ENTRY, or %LINE instruction.

User Action: Remove the illegal percent character from the instruction.

ILLPTHNAM **path name <text> is not a legal CDD name**

Severity: Error (E)

Explanation: A syntax message. The relative path name specified in the message is not a legal CDD name. For example:

- The name has more than 31 characters
- The first character is not alphabetic
- The remaining characters are not alphanumeric characters or a dollar sign ($) or an underscore (_)
- The last character is a dollar sign ($) or an underscore (_)

User Action: Make the relative path name conform to the rules for a legal CDD name.

ILLPRKNAM **illegal key name for PROGRAM KEY IS or DEFINE KEY AS**

Severity: Error (E)

Explanation: A mapping message. You specified a key in a PROGRAM KEY IS or DEFINE KEY AS instruction that is not one of the valid keys for the instruction.

User Action: Specify a key that is valid for the PROGRAM KEY IS or DEFINE KEY AS instruction.

ILLSLSHCHR **illegal slash character in text**

Severity: Error (E)

Explanation: A syntax message. A slash character appears as part of an instruction on the instruction line.

User Action: Remove the illegal slash character.

ILLSRCDAT **unsupported data type in source of mapping**

Severity: Explicit: Error (E)
%ALL: Information (I)

Index

In this index, a page number followed by a "t" indicates a table reference. A page number followed by an "f" indicates a figure reference.

B

C

D

E

F

H

I

K

L

O

P

Q

R

S

T

U

V

W

VAX TDMS
Reference Manual
AA-HU17A-TE
Including: AD-HU17A-T1

Reader's Comments

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement. _____

_____

_____

_____

Did you find errors in this manual? If so, specify the error and the page number.

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Do Not Tear - Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 33 MAYNARD MASS

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: DISG Documentation
ZK02-2/N53
Digital Equipment Corporation
110 Spit Brook Road
Nashua, NH 03062-2698

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line