VAX TDMS
Reference Manual

Order No. AA-HU17A-TE

August 1986

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

OPERATING SYSTEM: **VMS**
MicroVMS

SOFTWARE VERSION: **VAX TDMS V1.6**

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

digital™

Contents

Figures

Tables

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.

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 progamming 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 versions 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	TDMS error message codes that can be issued in a VAX DATATRIEVE application that uses TDMS.

Related Documents

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

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.
{ }	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.
. . .	Horizontal ellipses indicate that you can repeat the previous item one or more times.
. . .	Vertical ellipses in an example indicate that information unrelated to the example has been omitted.
( )	In RDU syntax, matching parentheses enclose lists of receiving fields in mapping instructions and CDD passwords.
UPPERCASE	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.

Technical Changes and New Features

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

- In addition to its regular date formats, TDMS now includes the standard VMS date format: DD-MMM-YYYY.

  The new date format is Date format 5 and was added to the Input and Output Mapping Tables.

- The default listing file name now contains up to 39 characters and can include the dollar sign ($) and underscore (_). This file name is generated in FDU by using the /OUTPUT qualifier and in RDU by using the /OUTPUT and /LIST qualifiers. The name is not truncated and includes any dollar signs and underscores.

- The LIST FORM command in FDU now lists all field attributes, including Must Fill and Response Required.

- Various errors in the Input and Output Mapping Tables have been corrected.

- You can now map signed numeric form fields to NL and NR record fields and not get data type conversion errors from RDU. The lengths of the form and record fields must be compatible.

  For more information, see Chapter 4 of the *VAX TDMS Request and Programming Manual.*

- TDMS has a predefined set of run-time function keys that operators can use to perform various operations on the screen such as moving from field to field, refreshing the screen, and getting help. These predefined keys are listed in Table 11-1 in the *VAX TDMS Request and Programming Manual* in the chapter called Program Request Keys.

- TDMS can now be used with VT200-series terminals set to VT200 mode. The following function keys from the LK201 keyboard used by VT200-series terminals are supported:

  - The F12 (BS) key performs the BACK SPACE key function.
  - The F13 (LF) key performs the LINE FEED key function.
  - The HELP key performs the PF2 or HELP key function.

  Information about these keys has been included where appropriate throughout the documentation set.

Form Definition Utility (FDU) Commands 1

This chapter provides information for all the commands in the Form Definition Utility (FDU). 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.
Notes	Provides information about using the command.
Examples	Gives examples on using the command.

1.1 Common FDU Qualifier, /AUDIT

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

The /AUDIT qualifier stores audit text with the form definition. The forms of the qualifier are:

/AUDIT

Includes the date and time you performed the specified command operation on the form definition and the name of the utility (FDU). /AUDIT is the default.

/NOAUDIT

Does not store audit text with the form definition. /AUDIT is the default.

/AUDIT = audit-string

Stores, with the form definition, an audit string that consists of one or more single words, quoted strings, text from a file, or a combination of these three items. The optional audit string can indicate, among other things, when the form 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, FDU 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 form 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 form 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 form definition, the text from the specified file or files. If you specify more than one file, enclose each @file-spec 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 form 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.

FDU 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.

1.2 @file-spec Command

Executes a command file that includes FDU commands.

Format

```
@file-spec
```

Prompts

```
FDU>
```

```
$
```

Command Parameter

file-spec

The name of the file containing a command procedure. The default file type is .COM.

Notes

The file can contain commands to process (CREATE, REPLACE, COPY, MODIFY) a form definition as well as other FDU commands.

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

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

Note

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

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

Examples

```
FDU> @PROCEDURE
```

Executes the commands in the file PROCEDURE.COM. For example, assume that PROCEDURE.COM contains:

```
SHOW VERSION
LIST FORM THISWEEK_PAYROLL/OUTPUT=[PAYROLL]JULY14.LIS
COPY FORM THISWEEK_PAYROLL LASTWEEK_PAYROLL
DELETE FORM THISWEEK_PAYROLL
EXIT
```

When you execute the command file, FDU:

- Displays the version of FDU on your terminal
- Writes a listing of the form definition to a file named JULY14.LIS in VMS directory [PAYROLL]
- Copies the form definition named THISWEEK_PAYROLL in your default CDD directory to a new form definition named LASTWEEK_PAYROLL
- Deletes the form definition THISWEEK_PAYROLL
- Exits from FDU

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

```
SET LOG
SET VERIFY
```

1.3 COPY FORM Command

Copies a form definition from one location in the CDD to another location.

Format

COPY FORM original-form-path-name new-form-path-name

Command Qualifiers	*Defaults*
/[NO]ACL	/ACL
/[NO]AUDIT /AUDIT = audit-string	/AUDIT /AUDIT
/[NO]LOG	/NOLOG

Prompts

```
FDU>
```

```
$
```

Command Parameters

original-form-path-name

The CDD path name (given, relative, or full) of an existing form.

new-form-path-name

The CDD path name (given, relative, or full) of the new location in the CDD to which the form is to be copied. The new name must not already exist.

Command Qualifiers

/ACL

Stores the form definition with a default CDD access control list (ACL). The ACL can grant or deny access to the form definition based on information contained in the list. For more information about the CDD access control list, refer to the VAX Common Data Dictionary documentation set. /ACL is the default.

/NOACL

Stores the form definition without an access control list. /ACL is the default.

/AUDIT

Stores audit text with the form definition. The standard default audit text includes the date and time you copy the form definition and the name of the utility (FDU). /AUDIT is the default.

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the form is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/LOG

Displays a message on the terminal indicating that FDU has successfully completed the operation. /NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that FDU has successfully completed the operation. /NOLOG is the default.

Note

The original form definition remains unchanged and in the original place. The new form definition may have the same given name but must not have the same full path name. The content of the new form definition is the same as the original except that the date and time stamp is updated.

Examples

```
FDU> COPY FORM ACCOUNTING.LOGIN_FORM PAYROLL.LOGIN_FORM
```

Copies the form definition ACCOUNTING.LOGIN_FORM in your default CDD directory to a form definition PAYROLL.LOGIN_FORM, also in your default CDD directory. The form in ACCOUNTING.LOGIN_FORM remains unchanged.

```
FDU> COPY FORM CDD$TOP.EXAMPLES.EMPLOYEE_FORM EMPLOYEE_FORM
```

Copies the form definition in CDD$TOP.EXAMPLES.EMPLOYEE_FORM to a form named EMPLOYEE_FORM in your default CDD directory. The form in CDD$TOP.EXAMPLES.EMPLOYEE_FORM remains unchanged.

1.4 CREATE FORM Command

Creates a new form definition in the CDD, either by using the TDMS form editor to generate a form or by using an existing FMS form file.

Format

```
CREATE FORM form-path-name

Command Qualifiers                    Defaults

/[NO]ACL                              /ACL

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

/FORM_FILE=file-spec                  None
/FORM_FILE=file-spec/V1               None

/[NO]LOG                              /NOLOG
```

Prompts

```
FDU>
```

```
$
```

Command Parameter

form-path-name

The CDD path name (given, relative, or full) of a form definition. The new name must not already exist.

Command Qualifiers

/ACL

Stores the form definition with a default CDD access control list. The ACL can grant or deny access to the form definition based on information contained in the list. For more information about the CDD access control list, refer to the VAX Common Data Dictionary documentation set. /ACL is the default.

/NOACL

Stores the form definition without an access control list. /ACL is the default.

/AUDIT

Stores audit text with the form definition. The standard default audit text includes the date and time you create the form definition and the name of the utility (FDU). /AUDIT is the default.

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the form is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/FORM_FILE=file-spec

The name of the file, generated using VAX FMS V2, that contains the form definition you want to store in the CDD. The default file type is .FRM. The file must contain a valid form.

/FORM_FILE=file-spec/V1

The name of the file, generated using the VAX FMS V1 form editor (FED), that contains the form definition you want to store in the CDD. The default file type is .FRM. The file must contain a valid form.

/LOG

Displays a message on the terminal indicating that FDU has successfully completed the operation. /NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that FDU has successfully completed the operation. /NOLOG is the default.

Notes

If you do not use the /FORM_FILE qualifier, the form editor creates a new form.

If you use the /FORM_FILE qualifier and the form definition includes any nonsupported features, those features are removed when the form definition is stored in the CDD, and FDU issues an informational message. When you use the /FORM_FILE qualifier, FDU stores the form without entering the form editor.

You cannot issue the CREATE FORM command from a batch file if you attempt to enter the form editor. If you issue the CREATE FORM command from a batch file, you must include the /FORM_FILE qualifier.

An error message is displayed in your output file or device defined as SYS$OUTPUT if you try to create a form that already exists in the CDD.

Examples

```
FDU> CREATE FORM FINANCE.LOGIN
```

Allows you to enter the form editor and define a new form that will be stored in the previously nonexistent location FINANCE.LOGIN in your default CDD directory.

```
FDU> CREATE FORM CDD$TOP.PAYROLL.SALARY_FORM-
FDU>_/AUDIT="Form for maintaining employee salary information"
```

Allows you to enter the form editor to define a new form that will be stored in CDD location CDD$TOP.PAYROLL.SALARY_FORM. When the form definition is created and stored in the CDD, the specified audit string is stored with the form definition.

```
FDU> CREATE FORM BILLING_FORM/AUDIT=(@TEXT1,@TEXT2,@TEXT3)
```

Allows you to enter the form editor to define a new form named BILLING_FORM that will be stored in your default CDD directory. When the form definition is created and stored in the CDD, the audit strings from the files TEXT1.DAT, TEXT2.DAT, and TEXT3.DAT are stored with the form definition.

```
FDU> CREATE FORM ACCOUNTS_FORM/AUDIT=@[JONES]TEXT4
```

Allows you to enter the form editor to define a new form named ACCOUNTS_FORM that will be stored in your default CDD directory. When the form definition is created and stored in the CDD, the audit string from the file TEXT4.DAT, located in directory JONES, is stored with the form definition.

1.5 CTRL/C Command

Cancels the current command but keeps you at FDU level.

Format

```
CTRL/C
```

Prompts

```
FDU>
```

```
$
```

Notes

Pressing CTRL/C at the FDU> prompt or at any point on the command line at the FDU> prompt cancels the current command line.

Pressing CTRL/C at a prompt for required parameters cancels the outstanding command.

Example

```
FDU> MODIFY FORM PAYROLL CTRL/C
Cancel

%FDU-E-CTRLCABORT, operation terminated with Control C
FDU>
```

FDU cancels the MODIFY FORM command, displays a message, and returns you to the FDU> prompt.

1.6 CTRL/Y Command

Causes an immediate exit from FDU.

Format

```
CTRL/Y
```

Prompts

```
FDU>
```

```
$
```

Note

Pressing CTRL/Y at any time causes an immediate exit from FDU, whether you are at FDU command level or in the form editor. If logging is enabled, the log file remains intact; however, it may not include the last command.

Example

```
FDU> CREATE CTRL/Y
Interrupt
```

```
$
```

FDU cancels the CREATE command, displays a message, and returns you to DCL level.

1.7 CTRL/Z Command

Causes FDU to stop accepting commands and to either return to DCL level or continue processing the previous command stream.

Format

```
CTRL/Z
```

Prompts

```
FDU>
```

```
$
```

Notes

Pressing CTRL/Z at the FDU> prompt returns you to DCL level.

Pressing CTRL/Z after entering a valid command at FDU level causes the command to be executed.

Pressing CTRL/Z while using the form editor results in an error.

Do not use CTRL/Z in either indirect command files or batch command files. In these cases, FDU (or DCL) attempts to process CTRL/Z as a command and returns an error message.

Examples

```
FDU> CTRL/Z
$
```

Entering CTRL/Z at FDU level returns you to DCL level.

```
FDU> MODIFY FORM BILLING_FORM CTRL/Z
```

When you press CTRL/Z, FDU stops taking instructions and checks the portion of the command entered prior to the CTRL/Z for errors. In this case, because the command is valid, FDU places BILLING_FORM in the form editor and displays it on the terminal. When you leave the form editor, you return to DCL level.

1.8 DELETE FORM Command

Deletes a form from the CDD.

Format

DELETE FORM form-path-name	
Command Qualifiers	*Defaults*
/[NO]CONFIRM	/NOCONFIRM
/[NO]LOG	/NOLOG

Prompts

```
FDU>
$
```

Command Parameter

form-path-name

The CDD path name (given, relative, or full) of an existing form definition.

Command Qualifiers

/CONFIRM

Instructs FDU to ask for confirmation before deleting the form definition. /NOCONFIRM is the default.

/NOCONFIRM

Instructs FDU to delete the form definition without asking for confirmation. /NOCONFIRM is the default.

/LOG

Displays a message on the terminal indicating that FDU has successfully completed the operation. /NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that FDU has successfully completed the operation. /NOLOG is the default.

Note

If the form does not exist, FDU issues an error message and returns you to the FDU> prompt or to DCL level.

Examples

```
FDU> DELETE FORM ACCOUNTING.LOGIN
```

Deletes the form ACCOUNTING.LOGIN in your default CDD directory.

```
FDU> DELETE FORM CDD$TOP.PAYROLL.SALARY_FORM/CONFIRM
Delete form CDD$TOP.PAYROLL.SALARY_FORM (y or n)? Y
FDU>
```

Asks you to confirm that you wish to delete the form definition SALARY_FORM from the CDD directory CDD$TOP.PAYROLL; if you do, it deletes the form and returns you to the FDU> prompt.

1.9 EDIT Command

Allows you to edit and resubmit the previous FDU command.

Format

```
EDIT
```

Prompts

```
FDU>
$
```

Notes

The EDIT command lets you edit, with a text editor, the last command that you entered at FDU level; it *does not* give direct access to the form editor. To understand how to use the form editor, see the CREATE FORM, MODIFY FORM, and REPLACE FORM commands.

You can type the EDIT command as soon as you invoke FDU at either DCL level or the FDU> prompt.

If you enter the EDIT command before typing an FDU command, FDU provides an edit buffer, and you can enter FDU commands using the features of your editor. You do not see the FDU> prompt or error messages as you enter the commands.

When you issue the EDIT command, TDMS executes the system-defined logical TDMS$EDIT, which invokes the system command procedure SYS$COMMON:[SYSEXE]TDMSEDIT.COM. That command procedure invokes the EDT editor and your EDT startup file EDTINI.EDT, if any exists.

You can change the FDU command EDIT to invoke a personal command procedure that invokes a particular editor or a particular set of editor startup commands. To do so, define the process logical FDU$EDIT to call a command procedure that you create, which in turn invokes the editor of your choice.

Example

```
FDU> CREATE FORM ACCOUNTING_FORM/FORM_FILE=[SMITH.FORMS]PAY.FRM
FDU> EDIT
```

In this sequence, the text editor is called and the previous command (CREATE FORM ...) is placed in the edit buffer. You can then change the name of the form file (for example, from PAY.FRM to PROJECT.FRM), and the new command is submitted to FDU when you leave the text editor.

1.10 EXIT Command

Causes FDU to stop accepting commands and to either return to DCL level or continue processing the previous command stream. It also causes an exit from a command file.

Format

```
EXIT
```

Prompt

```
FDU>
```

Note

FDU exits from either the utility or the command file.

Examples

```
FDU> EXIT
$
```

FDU stops taking commands and returns to DCL level.

```
FDU>@PROCEDURE
  SET DEFAULT CDD$TOP.EMPLOYEE
  SET LOG
  EXIT
FDU>
```

FDU stops taking commands from an indirect command file and returns to the FDU> prompt.

1.11 HELP Command

Provides online help for FDU commands.

Format

HELP [topic [subtopic]]	
Command Qualifier	*Default*
/[NO]PROMPT	/PROMPT

Prompts

```
FDU>
```

```
$
```

Command Parameters

topic

The FDU topic for which help is desired.

subtopic

The FDU parameters and qualifiers for which help is desired.

Command Qualifiers

/PROMPT

Prompts you for the topic and optionally, the subtopic for which you want help. Unlike all other FDU commands, you specify the HELP command qualifiers immediately after the command rather than at the end of the command line. The default is /PROMPT.

/NOPROMPT

Lets you obtain help on an FDU command without being prompted. Unlike all other FDU commands, you specify the HELP command qualifiers immediately after the command rather than at the end of the command line. The default is /PROMPT.

Notes

The form editor has its own Help facility. For help while using the form editor, press the HELP key (PF2 or F15).

When you issue the HELP command, you can obtain help on all the FDU commands, their qualifiers, and other FDU topics. To obtain a display of all the information under help in FDU, enter the following:

```
FDU> HELP *...
```

To obtain a hardcopy of the help information, you must assign SYS$OUTPUT at DCL level to be a line printer or other printing device. For example:

```
$ DEFINE SYS$OUTPUT LP:
$ FDU
FDU> HELP *...
```

The first command assigns the line printer to be SYS$OUTPUT; the second calls FDU; and the third requests a display of all help in FDU. The output is queued to the system printer.

To obtain a display of all TDMS help at DCL level, type the following:

```
$ HELP TDMS...
```

Example

```
FDU> HELP

  Information available:

  @           COPY       CREATE      DELETE      EDIT       EXIT
  form-editor            HELP        LIST        MODIFY     REPLACE     SAVE
  SET         SHOW       specify

Topic? CTRL/Z
FDU>
```

Enters help at FDU level.

1.12 LIST FORM Command

Provides a listing that shows background text, field data, and attribute information about the form definition. You can display the listing on a terminal or direct it to a file or line printer.

Format

LIST FORM form-path-name	
Command Qualifiers	*Defaults*
/OUTPUT[=file-spec]	/OUTPUT=SYS$OUTPUT
/[NO]PRINT	/NOPRINT

Prompts

```
FDU>
```

```
$
```

Command Parameter

form-path-name

The CDD path name (given, relative, or full) of an existing form definition.

Command Qualifiers

/OUTPUT[=file-spec]

Lists form definition information in a file with the specified file name.

If you do not specify a name following the /OUTPUT qualifier, FDU takes the given name of the full form definition path name up to 39 characters including dollar signs ($) and underscores (_). If that results in a legal file name, then FDU uses that file name with a file type .LIS. If not, FDU uses the file name FDULIS.LIS.

If you do specify a name following the /OUTPUT qualifier, then that name can be any standard VMS file name up to 39 characters long. An illegal file name will cause the command to fail.

The default is /OUTPUT=SYS$OUTPUT.

/PRINT

Prints a listing of the form definition on the system printer. If you use the /PRINT qualifier without the /OUTPUT qualifier, FDU creates a temporary listing file and deletes it after the print operation is completed. /NOPRINT is the default.

/NOPRINT

Does not print a listing of the form definition. /NOPRINT is the default.

Notes

If you do not use the /OUTPUT or the /PRINT qualifier, FDU writes the form listing to SYS$OUTPUT but creates no permanent listing file, unless SYS$OUTPUT is defined as a permanent file.

The information provided by the LIST FORM command is particularly useful when writing requests and records.

Examples

```
FDU> LIST FORM ACCOUNTING.PAYROLL
```

Lists all the information about the form definition ACCOUNTING.PAYROLL on your terminal (SYS$OUTPUT).

```
FDU> LIST FORM ACCOUNTING.PAYROLL/OUTPUT
```

Lists all information about the form definition ACCOUNTING.PAYROLL in a file named PAYROLL.LIS in your VMS default directory.

```
FDU> LIST FORM SALARY_FORM/OUTPUT=FORM16.XYZ/PRINT
```

Lists the information about form definition SALARY_FORM in a file named FORM16.XYZ and prints the information on the system printer.

1.13 MODIFY FORM Command

Allows you to edit an existing form definition with the form editor.

Format

MODIFY FORM form-path-name	
Command Qualifiers	*Defaults*
/[NO]AUDIT	/AUDIT
/AUDIT = audit-string	/AUDIT
/[NO]LOG	/NOLOG

Prompts

```
FDU>
```

```
$
```

Command Parameter

form-path-name

The CDD path name (given, relative, or full) of an existing form definition.

Command Qualifiers

/AUDIT

Stores audit text with the form definition. The standard default audit text includes the date and time you modify the form definition and the name of the utility (FDU). /AUDIT is the default.

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the form is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/LOG

Displays a message on the terminal indicating that FDU has successfully completed the operation. /NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that FDU has successfully completed the operation. /NOLOG is the default.

Notes

FDU places an existing form definition in the form editor. When you modify and save the form definition, the original form definition is deleted. If you choose not to save the modified form definition when you leave the form editor, the original remains in the CDD.

You cannot issue the MODIFY FORM command from a batch file.

Example

```
FDU> MODIFY FORM PAYROLL.LOGIN_FORM
```

Places the form definition PAYROLL.LOGIN_FORM, which is in your default CDD directory, in the form editor.

1.14 REPLACE FORM Command

Replaces a form definition in the CDD with a new form definition.

Format

REPLACE FORM form-path-name

Command Qualifiers	*Defaults*
/[NO]ACL	/ACL
/[NO]AUDIT	/AUDIT
/AUDIT = audit-string	/AUDIT
/[NO]CREATE	/CREATE
/FORM_FILE = file-spec	None
/FORM_FILE = file-spec/V1	None
/[NO]LOG	/NOLOG

Prompts

```
FDU>
```

```
$
```

Command Parameter

form-path-name

The CDD path name (given, relative, or full) of a form definition.

Command Qualifiers

/ACL

Stores the form definition with a default CDD access control list. The ACL can grant or deny access to the form definition based on information contained in the list. For more information about the CDD access control list, refer to the VAX Common Data Dictionary documentation set.

/ACL applies only when the REPLACE FORM command defaults to the CREATE FORM command. /ACL is the default.

/NOACL

Stores the form definition without an access control list. /ACL is the default.

/AUDIT

Stores audit text with the form definition. The standard default audit text includes the date and time you replace the form definition and the name of the utility (FDU). /AUDIT is the default.

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the form is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/CREATE

Enables FDU to treat the REPLACE FORM command as though it were the CREATE FORM command when it cannot find a form definition with the specified name in the CDD. /CREATE is the default.

/NOCREATE

Prevents FDU from treating the REPLACE FORM command as though it were the CREATE FORM command. Instead of creating a new form definition, FDU issues an error message and returns you to the FDU> prompt. /CREATE is the default.

/FORM_FILE=file-spec

The name of the file, generated using VAX FMS V2, that contains the form definition you want to store in the CDD. The default file type is .FRM. The file must contain a valid form.

/FORM_FILE=file-spec/V1

The name of the file, generated using the VAX FMS V1 form editor (FED), that contains the form definition you want to store in the CDD. The default file type is .FRM. The file must contain a valid form.

/LOG

Displays a message on the terminal indicating that FDU has successfully completed the operation. /NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that FDU has successfully completed the operation. /NOLOG is the default.

Notes

If a form definition of the name you specify does not already exist in the CDD, FDU treats the REPLACE FORM command as if it were the CREATE FORM command. If you want to be sure that you are replacing an existing form definition, use the /NOCREATE qualifier.

If you do not use the /FORM_FILE qualifier, FDU places you in the form editor buffer to create an entirely new form definition. No existing information is placed in the form editor buffer. If you save the new form definition in the Exit phase of the form editor, the original form definition is deleted. If you do not save the new form definition when you leave the form editor, the original form definition remains in the CDD.

You cannot issue the REPLACE FORM command from a batch file if you attempt to enter the form editor. If you issue the REPLACE FORM command from a batch file, you must include the /FORM_FILE qualifier.

When you use the /FORM_FILE qualifier, FDU stores the form without entering the form editor. If you use the /FORM_FILE qualifier and the form definition includes any features that TDMS does not support, those features are removed when the form definition is stored in the CDD. FDU issues an informational message indicating that the nonsupported features have been removed.

Example

```
FDU> REPLACE FORM ACCOUNTING.LOGIN
```

Calls the form editor. Because you are replacing a form, no existing information is placed in the form editor buffer. If you choose to save the new form definition (in the Exit phase of the form editor), it is stored in the CDD object ACCOUNTING.LOGIN. If you exit from the form editor without saving the form definition, the original form definition in ACCOUNTING.LOGIN remains unchanged.

1.15 SAVE Command

Saves in a file the most recent command you entered in FDU.

Format

```
SAVE file-spec
```

Prompt

```
FDU>
```

Command Parameter

file-spec

The name of the file in which you want to save the last FDU command. You may use any standard VMS file specification up to 39 characters in length. If you do not specify a directory, FDU files it in the directory in which you are currently working. The default file type is .SAV.

Example

```
FDU> CREATE FORM TESTFORM_FORM
        .
        .
        .
(Form displayed here)
        .
        .
        .
FDU> SAVE
Save to file      : TESTFORM.TXT
%FDU-I-SAVETOFIL, previous command SAVEd to file DUA1:[JONES.WORK]TESTFORM.TXT
FDU> EXIT

$ TY TESTFORM.TXT
CREATE FORM TESTFORM_FORM
```

FDU prompts you for a file name (if you do not specify it following the SAVE command). It writes the CREATE FORM command to the file you specify, TESTFORM.TXT, in the directory [JONES.WORK]. The example shows the file contents that FDU saves.

1.16 SET DEFAULT Command

Sets FDU to point to a CDD directory for as long as you are in FDU or until you set FDU to point to a new directory.

Format

```
SET DEFAULT CDD-directory-path-name
```

Prompt

```
FDU>
```

Command Parameter

CDD-directory-path-name

The CDD path name (given, relative, or full) of the default CDD directory.

Notes

Once you specify a default CDD directory, all path names are interpreted as starting at that level. You can override this default directory setting by specifying the full path name beginning with CDD$TOP.

The CDD default directory setting ends when you exit FDU. You must reset the CDD default directory when you reenter FDU.

If you want to create a CDD directory setting that exists every time you enter FDU or until you reset it, you can either:

- Enter the SET DEFAULT command in your startup command file FDUINI.COM. (See the @file-spec command.)
- Use the DCL DEFINE command to define the logical name CDD$DEFAULT in your login command file. (See Chapter 2 of the *VAX TDMS Forms Manual.*)

Examples

```
FDU> SET DEFAULT CDD$TOP.PERSONNEL.ACCOUNTING.SMITH
```

Sets the CDD default directory to
CDD$TOP.PERSONNEL.ACCOUNTING.SMITH

```
$ EDIT FDUINI.COM
  SET DEFAULT CDD$TOP.YOUR_HOME_DIRECTORY
  CTRL/Z
   *EXIT
$ FDU
FDU> SHOW DEFAULT
current CDD default path is 'CDD$TOP.YOUR_HOME_DIRECTORY'
```

Places the SET DEFAULT command in the startup command file that is executed each time you invoke FDU.

1.17 SET [NO]LOG Command

Creates a log and writes further output to a default or specified log file.

Format

```
SET [NO]LOG [file-spec]
```

Prompt

```
FDU>
```

Command Parameter

file-spec

The name of the file to which you want FDU to log. Use the standard VMS file specification.

If you specify the name previously given in the current FDU session, FDU creates a new version of that log file. If you specify a different name, FDU closes the first log file and opens the new one.

(If you do not specify a file, FDU logs to a default file specification as described in the Notes section.)

Notes

FDU logs the following data to the log file:

- An exact copy of each of the commands you enter or a copy of the commands executed by FDU from an indirect command file (if the indirect command file is executed in Set Verify mode)
- FDU messages preceded by an exclamation point (!) comment character

You can issue the SET LOG command without specifying a file name. FDU creates a default file in one of three ways:

- FDU creates a default log file by translating the logical name FDULOG.
- If the system does not have an equivalent name for FDULOG, FDU uses the file name FDULOG.LOG.

- If a current log file already exists when you issue the SET LOG command without naming a file, FDU issues a warning and continues to use the current log file.

The SET LOG command remains in effect for the entire session. If you exit FDU and reenter, you must issue the SET LOG command again. FDU opens a new log file.

The SET NOLOG command deactivates logging in FDU. SET NOLOG is the default. If you want SET LOG to be the default, you can include a SET LOG command in your FDUINI.COM startup command file.

Examples

```
FDU> SET LOG [SMITH.FORMS]FORMLOG
```

Enables logging and writes information about the commands issued and system responses to the file FORMLOG.LOG in the VMS directory [SMITH.FORMS]. (The default file type is .LOG)

```
FDU> SET NOLOG
```

Turns off logging. It is unnecessary to identify the file to which the information was logged.

```
$ ASSIGN [SMITH.FORMS]FORMLOG FDULOG
$ RUN SYS$SYSTEM:FDU
FDU> SET LOG
```

Assigns the logical name FDULOG to the file [SMITH.FORMS]FORMLOG.LOG, enters FDU, and enables logging. The log file is [SMITH.FORMS]FORMLOG.LOG.

1.18 SET [NO]VERIFY Command

Enables the display (on a terminal) or printing of the contents of FDU command procedures.

Format

```
SET [NO]VERIFY
```

Prompt

```
FDU>
```

Notes

By default, FDU is set to:

- Noverify in Interactive mode. When FDU executes commands from an indirect command file, it does not display those commands to SYS$OUTPUT.
- Verify in Batch mode. In Batch mode, FDU writes the command it is executing to the batch log file.

Note that the FDU default can be changed by a SET [NO]VERIFY command in either the command procedure being invoked or your FDUINI.COM file.

Examples

```
FDU> SET VERIFY
FDU> @PROCEDURE
```

As FDU executes the commands in the file PROCEDURE.COM, the commands are displayed on the terminal (or SYS$OUTPUT).

```
FDU> SET NOVERIFY
FDU> @ACCTREQ
```

Places FDU in Noverify mode. When you pass the command file ACCTREQ.COM to FDU, it does not display the commands it executes from that file.

1.19 SHOW DEFAULT Command

Displays the current CDD default directory.

Format

```
SHOW DEFAULT
```

Prompts

```
FDU>
```

```
$
```

Note

Shows the current CDD default directory. You can define the default in FDU using the SET DEFAULT command or, at DCL level using the DEFINE command, to define CDD$DEFAULT to point to the desired CDD directory.

Example

```
FDU> SHOW DEFAULT
current CDD default path is 'CDD$TOP.PERSONNEL'
FDU>
```

Indicates that the current default directory setting is CDD$TOP.PERSONNEL. When you process (create, list, etc.) form definitions using the given name or a relative path name, FDU looks for or stores the definitions in CDD$TOP.PERSONNEL.

1.20 SHOW LOG Command

Displays information about the current logging status of FDU.

Format

```
SHOW LOG
```

Prompt

```
FDU>
```

Note

The SHOW LOG command tells whether logging is taking place, and, if so, the name of the file to which the log is written. If logging is not taking place, FDU indicates that logging is not taking place to FDULOG.

Examples

```
FDU> SHOW LOG
not logging to file FDULOG
```

FDU indicates no logging is taking place.

```
FDU> SET LOG
FDU> SHOW LOG
logging to file DBA1:[JONES]FDULOG.LOG;1
FDU>
```

FDU indicates it is logging to default FDULOG.LOG in the current VMS directory.

1.21 SHOW VERSION Command

Displays the current version of FDU.

Format

```
SHOW VERSION
```

Prompts

```
FDU>
```

```
$
```

Example

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

FDU shows the version of 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.

2.2 @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

2.3 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.

Specify the request library file name only if the CDD request library definition does not contain a FILE IS instruction.

If you specify both the request-library-file parameter and a FILE IS instruction in a request library definition, the request-library-file parameter takes precedence.

Command Qualifiers

/AUDIT

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

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the request library definition is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/LIST

Creates a listing file in your default or specified VMS directory that contains:

- Information about the build operation
- Incorrect request instructions and warning or error level messages resulting from the build operation

When the /LIST qualifier is used with the /LOG qualifier, the listing file contains information and warning level messages indicating the %ALL mappings that were created and those that were not created and why. (An A in the left column of a listing line indicates that the line gives information about a %ALL mapping.)

If you do not specify a name following the /LIST qualifier, RDU takes the given name of the full request library definition path name up to 39 characters including dollar signs ($) and underscores (_). If that results in a legal file name, then RDU uses that file name with a file type of .LIS. If not, RDU uses the file name RDULIS.LIS.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/NOLIST

Specifies that no listing file be created.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LIST=file-spec

Creates a listing file with the given file specification. You may use any standard VMS file name up to 39 characters in length. An illegal file name will cause the command to fail.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LOG

Displays on the terminal screen messages that indicate:

- The request names and the form and record definition names that are referred to in the request library definition
- Whether or not the operation successfully completed

In addition, if the request was created with the /NOSTORE qualifier, or if the request changed since the request binary was created, the following messages are displayed:

- The mappings that are correct for all the %ALL mapping instructions in the request
- The mappings that are incorrect for all the %ALL mapping instructions in the request

No matter what the /LOG setting, RDU displays explicit mappings that are incorrect and the associated error messages. RDU does not generate messages about the individual explicit mappings that are correct.

/NOLOG is the default.

/NOLOG

Displays no messages on the terminal indicating RDU has successfully completed the operation. Also does not display information about request names and form and record definition names or mapping messages. /NOLOG is the default.

/PRINT

Prints a listing of the request library definition on the system printer. If not used with the /LIST qualifier, /PRINT creates a temporary listing file and deletes it after printing. /NOPRINT is the default.

/NOPRINT

Does not print a listing of the request library definition on the system printer. /NOPRINT is the default.

Notes

If the request has an associated request binary structure, RDU checks that any related forms and records have not changed. If they have not changed, the request binary structure is written to the request library file.

If the request does not have an associated request binary structure, or if any related forms or records have changed, RDU builds the request binary structure and writes it to the request library file.

RDU does not build a request library file if:

- The requests named in the request library definition are not in the CDD.
- You did not specify a request library file name.
- The form names and record names referred to in the requests are not in the CDD.
- The form field names explicitly specified in the requests do not match the form field names in the CDD form definitions.
- The record field names explicitly specified in the requests do not match the record field names in the CDD record definitions.

- The explicit mappings in the requests are incorrect. The mappings are incorrect if:
  - The data types and lengths of fields you map to each other are not compatible
  - The structures of the form and record fields you map are not compatible

In the case of incorrect %ALL mappings or field references, RDU builds the request library file unless all the mappings implied by a %ALL instruction are incorrect.

If the BUILD LIBRARY command fails, RDU:

- Returns an error message to tell you which reference or mapping is incorrect
- Returns you to the RDU> prompt (or the DCL prompt if you entered the BUILD LIBRARY command at DCL level)
- Does not attempt to build a request library file

You must correct mappings or references before a BUILD command can succeed.

Examples

```
RDU> BUILD LIBRARY CDD$TOP.SAMPLE.EMPLOYEE_LIBRARY
```

Builds a request library file using the requests listed in the CDD request library definition EMPLOYEE_LIBRARY. This request library is in the CDD directory CDD$TOP.SAMPLE.

The name of the request library file is not shown. It is contained in the CDD request library.

```
RDU> BUILD LIBRARY SAMPLE.EMPLOYEE_LIBRARY   EMPLOYEE.RLB
```

Builds a request library file using the requests named in the CDD request library EMPLOYEE_LIBRARY.

The request library file, EMPLOYEE.RLB, is specified as a parameter to the BUILD LIBRARY command.

```
$ RDU BUILD LIBRARY PERSONNEL_LIB PERSON.RLB/LOG
```

Builds a request library file, PERSON.RLB, and displays a success level message indicating the library file has been built and shows the mappings RDU placed in the file. Note that the command is entered at DCL level.

2.4 COPY LIBRARY Command

Copies a request library definition from one location in the CDD to another location.

Format

```
COPY LIBRARY original-request-library-path-name
             new-request-library-path-name
```

Command Qualifiers	*Defaults*
/[NO]ACL	/ACL
/[NO]AUDIT	/AUDIT
/AUDIT = audit-string	/AUDIT
/[NO]LOG	/NOLOG

Prompts

```
RDU>
```

```
$
```

Command Parameters

original-request-library-path-name

The CDD path name (given, relative, or full) of an existing request library definition.

new-request-library-path-name

The CDD path name (given, relative, or full) of the new location in the CDD to which RDU is copying the request library definition. The new name must not already exist.

Command Qualifiers

/ACL

Stores the request library definition with a default CDD access control list (ACL). The ACL can grant or deny access to the request library definition based on information contained in the list. For more information about the CDD access control list, refer to the VAX Common Data Dictionary documentation set. /ACL is the default.

/NOACL

Stores the request library definition without an access control list. /ACL is the default.

/AUDIT

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

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the request library definition is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/LOG

Displays a message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

Note

The original request library definition remains unchanged and in the original place. The new request library definition may have the same given name but must not have the same full path name. The content of the new request library definition is the same as the original except that the date and time stamp is updated.

Examples

```
RDU>  COPY LIBRARY EMPLOYEE_LIBRARY -
RDU>_CDD$TOP.ACCOUNTING.PERSONNEL.EMPLOYEE_LIBRARY
```

Copies the request library definition EMPLOYEE_LIBRARY from your default CDD directory to the CDD directory CDD$TOP.ACCOUNTING.PERSONNEL. The given names of both request library definitions remain the same.

```
RDU>  COPY LIBRARY CDD$TOP.SAMPLE.EMPLOYEE_BASIC -
RDU>_ CDD$TOP.PERSONNEL.EMPLOYEE_BASIC
```

Copies the request library definition EMPLOYEE_BASIC from CDD$TOP.SAMPLE to directory CDD$TOP.PERSONNEL. The given names remain the same.

```
$ RDU  COPY LIBRARY PERSONNEL_ADD_LIBRARY  -
$_ PERSONNEL_CHANGE_LIBRARY
```

Copies the request library definition PERSONNEL_ADD_LIBRARY to the new library definition PERSONNEL_CHANGE_LIBRARY. Both library definitions are in the same CDD directory. The given names are different. Note that you must type the symbol RDU to enter the COPY LIBRARY command at DCL level.

2.5 COPY REQUEST Command

Copies a request from one location in the CDD to another location.

Format

COPY REQUEST original-request-path-name new-request-path-name

Command Qualifiers	*Defaults*
/[NO]ACL	/ACL
/[NO]AUDIT	/AUDIT
/AUDIT = audit-string	/AUDIT
/[NO]LOG	/NOLOG

Prompts

```
RDU>
```

```
$
```

Command Parameters

original-request-path-name

The CDD path name (given, relative, or full) of an existing request.

new-request-path-name

The path name (given, relative, or full) of the new location in the CDD to which RDU is copying the request. The new name must not already exist.

Command Qualifiers

/ACL

Stores the request with a default CDD access control list. The ACL can grant or deny access to the request based on information contained in the list. For more information about the CDD access control list, refer to the VAX Common Data Dictionary documentation set. /ACL is the default.

/NOACL

Stores the request without an access control list. /ACL is the default.

/AUDIT

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

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the request is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/LOG

Displays a message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that RDU has successfully completed the operation.

Note

The original request remains unchanged and in the original place. The new request can have the same given name but must not have the same full path name. The content of the new request is the same as the original except that the date and time stamp is updated. Any associated request binary structure is also copied.

Examples

```
RDU>  COPY REQUEST EMPLOYEE_REQUEST -
RDU>_ CDD$TOP.ACCOUNTING.PERSONNEL.EMPLOYEE_REQUEST
```

Copies the request EMPLOYEE_REQUEST from your default CDD directory to the CDD directory CDD$TOP.ACCOUNTING.PERSONNEL. The given name of both requests is the same.

```
RDU>  COPY REQUEST CDD$TOP.SAMPLE.EMPLOYEE_BASIC -
RDU>_ CDD$TOP.PERSONNEL.EMPLOYEE_BASIC
```

Copies the request EMPLOYEE_BASIC from the CDD$TOP.SAMPLE CDD directory to the CDD$TOP.PERSONNEL CDD directory. The given names of both requests are the same.

```
$  RDU COPY REQUEST PERSONNEL_ADD_REQUEST -
$_  PERSONNEL_CHANGE_REQUEST
```

Copies the request PERSONNEL_ADD_REQUEST to the request PERSONNEL_CHANGE_REQUEST in the same CDD directory. Note that you must type the symbol RDU when you enter RDU commands at DCL level.

2.6 CREATE LIBRARY Command

Allows you to enter request library definition text in RDU and, if it is error-free, stores the request library definition in the CDD.

Format

CREATE LIBRARY request-library-path-name [file-spec]

Command Qualifiers	*Defaults*
/[NO]ACL	/ACL
/[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 you want to create.

file-spec

The text file containing the request library definition text that you pass to RDU to check and store in the CDD. If you do not specify a file type, RDU looks for the default file type .LDF.

If you do not specify a file-spec parameter, RDU expects you to enter the request library definition text from the default file or device defined as SYS$INPUT.

- In Interactive mode, the default is the terminal. You enter the text at the RDUDFN> prompt.
- In Batch mode, the default is the current command stream.

Command Qualifiers

/ACL

Stores the request library definition with a default CDD access control list. The ACL can grant or deny access to the request library definition based on information contained in the list. For more information about the CDD access control list, refer to the VAX Common Data Dictionary documentation set. /ACL is the default.

/NOACL

Stores the request library definition without an access control list. /ACL is the default.

/AUDIT

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

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the request library definition is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/LIST

Creates a listing file in your default or specified VMS directory. The listing file contains the request library definition text and error messages generated by RDU.

If you do not specify a name following the /LIST qualifier, RDU takes the given name of the full request library definition path name up to 39 characters including dollar signs ($) and underscores (_). If that results in a legal file name, then RDU uses that file name with a file type of .LIS. If not, RDU uses the file name RDULIS.LIS.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/NOLIST

Specifies that no listing file be created. /LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LIST=file-spec

Creates a listing file in your default or specified VMS directory with the given file specification. You may use any standard VMS file name up to 39 characters in length. An illegal file name will cause the command to fail.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LOG

Displays a message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

/PRINT

Prints a listing of the request library definition on the system printer. If not used with the /LIST qualifier, /PRINT creates a temporary listing file and deletes it after printing. /NOPRINT is the default.

/NOPRINT

Does not print a listing of the request library definition on the system printer. /NOPRINT is the default.

Notes

You can enter the request library definition text in one of two ways:

- Type it directly into RDU following the CREATE LIBRARY command.
- Type it into a text or command file and pass the file to RDU either at DCL level or at the RDU> prompt.

RDU does not create the request library definition in the CDD if:

- The CDD already contains a request library definition or other CDD object with the same path name
- RDU is in Validate mode and the request library definition specifies requests that are not in the CDD
- RDU finds syntax errors in the request library definition text

Instead, after checking the request library definition text and writing any errors to SYS$OUTPUT, RDU returns to the RDU> prompt (or the DCL prompt if the command was entered at DCL level).

If RDU is in Novalidate mode, it creates the request library definition without checking for the requests at the CDD locations you specify.

Examples

```
RDU> CREATE LIBRARY CDD$TOP.EMPLOYEE_DIR.PERSONNEL_EMPLOYEE
```

Creates the request library definition PERSONNEL_EMPLOYEE in the CDD directory CDD$TOP.EMPLOYEE_DIR.

```
RDU> CREATE LIBRARY
COMPONENT-NAME: CDD$TOP.SAMPLE_DIR.PERSONNEL_EMPLOYEE
```

Creates the request library definition PERSONNEL_EMPLOYEE in the directory CDD$TOP.SAMPLE_DIR.

```
$  RDU> CREATE LIBRARY YOUR_NAME_DIR.STATUS_LIBRARY -
$_ DBA1:[PERSONNEL.DIR]EMPLOYEE.DAT
```

Creates the request library definition STATUS_LIBRARY in the CDD directory YOUR_NAME_DIR by checking and storing the request library definition text contained in the file EMPLOYEE.DAT. Note that when you enter the CREATE LIBRARY command at DCL level, you must type the symbol RDU.

```
$ RDU @CRELIBEMP.COM
```

Executes the CREATE command in the command procedure file CRELIBEMP.COM and creates and stores the request library definition contained in that command file.

```
RDU> CREATE LIBRARY PERSONNEL_EMPLOYEE/LIST-
RDU>_=EMPLOYEE.LIS
```

Creates the request library definition PERSONNEL_EMPLOYEE and generates a listing file named EMPLOYEE.LIS in the default VMS directory.

2.7 CREATE REQUEST Command

Allows you to enter request instructions in RDU and, if they are error free, creates a request in the CDD. If RDU is in Store mode, the CREATE REQUEST command also creates a request binary structure and stores it in the CDD with the request.

Format

CREATE REQUEST request-path-name [file-spec]

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

Prompts

```
RDU>
```

```
$
```

Command Parameters

request-path-name

A CDD path name (given, relative, or full) of the request that you want to create.

file-spec

The file containing the request text that you pass to RDU to check and store in the CDD. If you do not specify a file type, RDU looks for the default file type .RDF.

If you do not specify a file-spec parameter, RDU expects you to enter the request text from the default file or device defined as SYS$INPUT:

- In Interactive mode, the default is the terminal. You enter the text at the RDUDFN> prompt.
- In Batch mode, the default is the current command stream.

Command Qualifiers

/ACL

Stores the request with a default CDD access contrc' list. The ACL can grant or deny access to the request based on information contained in the list. For more information about the CDD access control list, refer to the VAX Common Data Dictionary documentation set. /ACL is the default.

/NOACL

Stores the request without an access control list. /ACL is the default.

/AUDIT

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

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the request is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/LIST

Creates a listing file in your default or specified VMS directory. The listing file contains the request text and error messages generated by RDU.

When used with the /LOG qualifier, the /LIST qualifier creates a listing file that contains information and warning level messages indicating the %ALL mappings that were created and those that were not created and why. (An A in the left column of a listing line indicates that the line gives information about a %ALL mapping.)

If you do not specify a name following the /LIST qualifier, RDU takes the given name of the full request path name up to 39 characters including dollar signs ($) and underscores (_). If that results in a legal file name, then RDU uses that file name with a file type of .LIS. If not, RDU uses the file name RDULIS.LIS.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/NOLIST

Specifies that no listing file be created. /LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LIST=file-spec

Creates a listing file in your default or specified VMS directory with the given file specification. You may use any standard VMS file name up to 39 characters in length. An illegal file name will cause the command to fail.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LOG

Displays on the terminal screen a message that indicates that RDU has successfully completed the operation. If the request is created while in Validate mode, it also displays messages that indicate:

- The request names and the form and record definition names that RDU uses to build the request library file
- The mappings RDU creates for %ALL
- The mappings RDU does not create for %ALL

/NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that RDU has successfully completed the operation. Also does not display information about request names and form and record definition names or mapping messages. /NOLOG is the default.

/PRINT

Prints a listing of the request on the system printer. If not used with the /LIST qualifier, /PRINT creates a temporary listing file and deletes it after printing. /NOPRINT is the default.

/NOPRINT

Does not print a listing of the request on the system printer. /NOPRINT is the default.

/STORE

Stores the request binary structure in the CDD. /STORE is the default in Validate mode.

/NOSTORE

Does not store the request binary structure in the CDD. /STORE is the default in Validate mode.

Notes

You can enter the request in one of two ways:

- Type it directly into the RDU following the CREATE REQUEST command.
- Type it into a text or command file and pass the file to RDU either at DCL level or at the RDU> prompt.

If RDU is in Validate mode and the /NOSTORE qualifier is not used, the request binary structure is stored in the CDD with the request.

RDU does not create the request in the CDD if:

- The CDD already contains a request or any other CDD object with the same path name
- RDU finds syntax errors in the request

- RDU is in Validate mode and the request:
  - Refers to form and/or record definitions that do not exist in the CDD
  - Contains incorrect mapping instructions

Instead, after checking the request text and writing syntax or mapping error messages to SYS$OUTPUT, RDU returns you to the RDU> prompt (or the DCL prompt if you entered the CREATE REQUEST command at DCL level). In RDU, you can use the EDIT or SAVE command to correct or save the request text or commands.

If RDU is in Novalidate mode, it creates the request without checking for valid mapping references.

The default for the /[NO]STORE qualifier depends on the Validate mode. If SET VALIDATE is in effect, the default is /STORE. If SET NOVALIDATE is in effect, the default is /NOSTORE. If the request binary structure is stored with the request, then the BUILD LIBRARY command revalidates the request only if an associated form or record has changed since the request binary structure was created.

Examples

In all of the following examples, the request binary structure is stored with the request in the CDD if RDU is in Store mode.

```
RDU> CREATE REQUEST CDD$TOP.EMPLOYEE_DIR.PERSONNEL_EMPLOYEE
```

Creates the request PERSONNEL_EMPLOYEE in the CDD directory CDD$TOP.EMPLOYEE_DIR.

```
RDU> CREATE REQUEST
COMPONENT-NAME: CDD$TOP.SAMPLE_DIR.PERSONNEL_EMPLOYEE
```

Creates the request PERSONNEL_EMPLOYEE in the CDD directory CDD$TOP.SAMPLE_DIR.

```
$  RDU CREATE REQUEST YOUR_NAME_DIR.STATUS_REQUEST -
$_ DBA1:[PERSONNEL.DIR]EMPLOYEE.DAT
```

Creates the request STATUS_REQUEST in the CDD directory YOUR_NAME_DIR by translating and storing the request text contained in the EMPLOYEE.DAT file.

```
$ RDU @CREREQEMP.COM
```

Executes the CREATE REQUEST command in the command procedure file CREREQEMP.COM and creates and stores the request contained in that command file. Note that you type the RDU symbol when you enter any RDU command at DCL level.

```
RDU> CREATE REQUEST PERSONNEL_EMPLOYEE/LIST=EMPLOYEE.LIS
```

Creates the request PERSONNEL_EMPLOYEE and generates a listing file named EMPLOYEE.LIS in the default directory.

2.8 CTRL/C Command

Cancels the current command but remains at the current command level (either RDU or DCL).

Format

```
CTRL/C
```

Prompts

```
RDU>

$

RDUDFN>
```

Note

Entering CTRL/C outside of an editor cancels the current command and prompts for additional command input.

Example

```
RDU>   CREATE REQUEST  CTRL/C
%RDU-E-CTRLCABORT, operation terminated with Control C
RDU>
```

RDU cancels the CREATE REQUEST command, displays a message, and returns you to the RDU> prompt.

2.9 CTRL/Y Command

Causes an immediate exit from RDU.

Format

```
CTRL/Y
```

Prompts

```
RDU>

$

RDUDFN>
```

Note

If you press CTRL/Y while the SET LOG command is enabled, the log file may or may not contain the last command and the associated request or request library definition text.

Example

```
RDU> CREATE  (CTRL/Y)
$
```

RDU cancels the current command, exits RDU, and returns you to the DCL prompt.

2.10 CTRL/Z Command

Exits from RDU or RDUDFN and returns to the previous command stream.

Format

```
CTRL/Z
```

Prompts

```
RDU>

$

RDUDFN>
```

Notes

RDU exits from the current command stream (RDU, RDUDFN) and reverts to the previous command stream or exits the program.

Do not use the CTRL/Z command in either indirect command files or batch command files. In these cases, RDU (or DCL) attempts to process CTRL/Z as a command and returns an error message.

Examples

```
RDU> CREATE REQUEST STATUS_REQUEST
RDUDFN> FORM IS EMPLOYEE_FORM;
RDUDFN> RECORD IS EMPLOYEE_RECORD;
RDUDFN> CLEAR SCREEN; CTRL/Z
  0003              CLEAR SCREEN;
.............................1
%RDU-E-SYNTAXERR, Found end-of-file when expecting TDMS-Keyword
%RDU-E-ERRDPAR, Error during instruction processing
%RDU-E-NOREQCRE, no request created
RDU>
```

RDU stops taking instructions, exits from the RDUDFN> prompt, and checks the portion of the request that you entered prior to the CTRL/Z for errors. Since the partial request text is not valid (it lacks an END DEFINITION instruction), RDU issues error messages and returns you to command (RDU>) level.

CTRL/Z

```
$ RDU CTRL/Z
$
```

Entering CTRL/Z at DCL level returns you to DCL level.

2.11 DELETE LIBRARY Command

Deletes a request library definition from the CDD.

Format

DELETE LIBRARY request-library-path-name

Command Qualifiers	*Defaults*
/[NO]CONFIRM	/NOCONFIRM
/[NO]LOG	/NOLOG

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 delete from the CDD.

Command Qualifiers

/CONFIRM

Instructs RDU to ask for confirmation before it deletes the request library definition. /NOCONFIRM is the default.

/NOCONFIRM

Allows RDU to delete the request library definition without asking for confirmation. /NOCONFIRM is the default.

/LOG

Displays a message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

Note

If the request library definition named does not exist, RDU issues an error message and returns you to the RDU> or DCL prompt.

Examples

```
RDU> DELETE LIBRARY ACCOUNTING.EMPLOYEE
```

Deletes the request library definition EMPLOYEE from a CDD directory. RDU determines which directory it should delete the library from by appending the CDD directory ACCOUNTING to your CDD default directory.

```
$ RDU  DELETE LIBRARY PERSON_EMPLOYEE/CONFIRM/LOG
Delete library CDD$TOP.TEST.PERSON_EMPLOYEE (y or n)? Y

%RDU-S-RLBDELETE, library CDD$TOP.TEST.PERSON_EMPLOYEE deleted
```

Deletes the CDD directory PERSON_EMPLOYEE from the CDD directory CDD$TOP.TEST.

2.12 DELETE REQUEST Command

Deletes a request from the CDD.

Format

DELETE REQUEST request-path-name	
Command Qualifiers	*Defaults*
/[NO]CONFIRM	/NOCONFIRM
/[NO]LOG	/NOLOG

Prompts

```
RDU>
```

```
$
```

Command Parameter

request-path-name

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

Command Qualifiers

/CONFIRM

Instructs RDU to ask for confirmation before it deletes the request. /NOCONFIRM is the default.

/NOCONFIRM

Allows RDU to delete the request without asking for confirmation. /NOCONFIRM is the default.

/LOG

Displays a message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

Notes

If the request does not exist, RDU issues an error message and returns you to the RDU> or DCL prompt.

Deleting a request also deletes the binary structure associated with the request.

Examples

```
RDU> DELETE REQUEST ACCOUNTING.EMPLOYEE
```

Deletes the request EMPLOYEE from a CDD directory. RDU identifies the CDD directory by adding the CDD directory ACCOUNTING to your default CDD directory name.

```
$ RDU DELETE REQUEST PERSON_EMPLOYEE/CONFIRM/LOG
Delete request CDD$TOP.SAMPLE.PERSON_EMPLOYEE (y or n)? Y

%RDU-S-REQDELETE, request CDD$TOP.SAMPLE.PERSON_EMPLOYEE deleted
```

Deletes the request PERSON_EMPLOYEE from the CDD directory CDD$TOP.SAMPLE when you respond by typing Y to the confirm prompt. RDU displays a success message indicating the request is deleted.

2.13 EDIT Command

Allows you to edit the most recent command and associated request or request library text that you entered in RDU. Also allows you to create an edit buffer into which you can enter RDU commands and request or request library text.

Format

```
EDIT
```

Prompts

```
RDU>
```

```
$
```

Notes

You can use the EDIT command in two ways:

- Within RDU, you can issue the EDIT command *after* you enter an RDU command (such as CREATE REQUEST, REPLACE REQUEST, and so on) and the associated request or request library definition text.

  RDU places the command and all the text you entered following the command in the edit buffer. You can correct or change the RDU command or the text using your editor's features.

- You can enter the EDIT command as soon as you invoke RDU at either the DCL level or the RDU> prompt.

  If you enter the EDIT command before typing an RDU command, RDU provides an edit buffer, and you can enter RDU commands and request or request library text using the features of your editor. You do not see the RDU> prompts or error messages as you enter the text.

When you exit the editor, RDU executes the commands in the file.

If you quit the editing session without creating an output file, RDU returns you to the RDU> prompt (or the DCL prompt if you are at DCL level) and discards the request or request library definition text.

When you issue the EDIT command, TDMS executes the system-defined logical TDMS$EDIT, which invokes the system command procedure SYS$COMMON:[SYSEXE]TDMSEDIT.COM. That command procedure invokes the EDT editor and your EDT startup file EDTINI.EDT, if any exists.

You can change the RDU command EDIT to invoke a personal command procedure that invokes a particular editor or a particular set of editor startup commands. To do so, define the process logical RDU$EDIT to call a command procedure that you create, which in turn invokes the editor of your choice.

Examples

```
RDU> CREATE REQUEST EMPLOYEE_REQUEST
RDUDFN>     FORM IS EMPLOYEE_FORM;
RDUDFN>     DISPLAY FORM EMPLOYEE_FORM
RDUDFN>     END DEFINITION;
  0003              END DEFINITION;
.....................1
%RDU-E-NOSEMICLN, Missing ';' at end of previous instruction
%RDU-E-ERRDPAR, Error during instruction processing
%RDU-E-NOREQCRE, no request created
RDU> EDIT
     CREATE REQUEST EMPLOYEE_REQUEST
          FORM IS EMPLOYEE_FORM;
          DISPLAY FORM EMPLOYEE_FORM
          END DEFINITION;
CTRL/Z
*SUB/EMPLOYEE_FORM/EMPLOYEE_FORM;/ 3
3         DISPLAY FORM EMPLOYEE_FORM;
 1 substitution
* EXIT
RDU>
```

RDU puts the last command, CREATE REQUEST EMPLOYEE_REQUEST, and the associated request text in an edit buffer so you can edit the request or command using the features of your editor. When you exit the editor, RDU checks the request text for errors and creates a request in the CDD.

```
$  RDU EDIT
CREATE REQUEST EMPLOYEE_REQUEST/LOG
FORM IS EMPLOYEE_FORM;
RECORD IS EMPLOYEE_REC;
DISPLAY FORM EMPLOYEE_FORM;
END DEFINITION;
CTRL/Z
* EXIT
$
```

The EDIT command is entered at the DCL prompt. RDU provides an edit buffer into which you can enter the RDU commands and request text. When you exit the file, RDU executes the commands in the file.

2.14 EXIT Command

Causes RDU to stop accepting commands and either return to DCL level or continue processing the previous command stream. It also causes an exit from a command file.

Format

```
EXIT
```

Prompt

```
RDU>
```

Note

RDU exits from either the utility or the command file.

Examples

```
RDU> EXIT
$
```

RDU stops taking commands and returns to the DCL prompt.

```
RDU>  @EMPREQ
  SET VALIDATE !These lines are in the file EMPREQ.COM
  SET LOG
  EXIT
RDU>
```

RDU stops taking commands from an indirect file and returns to the RDU> prompt.

2.15 HELP Command

Provides online help for RDU commands.

Format

HELP [topic [subtopic]]	
Command Qualifier	*Default*
/[NO]PROMPT	/PROMPT

Prompts

```
RDU>
```

```
$
```

Command Parameters

topic

The RDU subject for which you want help.

subtopic

The RDU parameter or qualifier for which you want help.

Command Qualifiers

/PROMPT

Prompts you for the topic and optionally the subtopic for which you want help. Unlike all other RDU commands, you specify the HELP command qualifiers immediately after the command rather than at the end of the command line. /PROMPT is the default.

/NOPROMPT

Lets you obtain help on an RDU command without being prompted. Unlike all other RDU commands, you specify the HELP command qualifiers immediately after the command rather than at the end of the command line. /PROMPT is the default.

Notes

When you issue the HELP command, you can obtain help on all the RDU commands, their qualifiers, and other RDU topics.

To obtain a display of all the information under help in RDU, enter the following:

```
RDU> HELP *...
```

To obtain a hardcopy of the help information, you must assign SYS$OUTPUT to be a line printer or other printing device at DCL level. For example:

```
$ DEFINE SYS$OUTPUT LP:
$ RDU
RDU> HELP *...
```

The first command assigns the line printer to be SYS$OUTPUT; the second calls RDU; and the third requests a display of all help in RDU. The output is queued to the system printer.

To obtain a display of all TDMS help at VMS level, type the following:

```
$ HELP TDMS...
```

Example

```
RDU> HELP

  Information available:
  @         BUILD    COPY     CREATE     DELETE     EDIT     EXIT
  HELP      LIST     MODIFY   REPLACE    SAVE       SET      SHOW
  specify   VALIDATE

Topic? CTRL/Z
RDU>
```

Enters Help at RDU level.

2.16 LIST LIBRARY Command

Provides a listing that shows the source text of the request library definition. You can display the listing on a terminal or direct it to a file or line printer.

Format

LIST LIBRARY request-library-path-name	
Command Qualifiers	*Defaults*
/OUTPUT[=file-spec]	/OUTPUT=SYS$OUTPUT
/[NO]PRINT	/NOPRINT

Prompts

```
RDU>
$
```

Command Parameter

request-library-path-name

The CDD path name (given, relative, or full) of the request library definition to be listed.

Command Qualifiers

/OUTPUT[=file-spec]

Lists request library definition information to a file with the specified file name.

If you do not specify a name following the /OUTPUT qualifier, RDU takes the given name of the full request library definition path name up to 39 characters including dollar signs ($) and underscores (_). If that results in a legal file name, then RDU uses that file name with a file type of .LIS. If not, RDU uses the file name RDULIS.LIS.

If you specify a name following the /OUTPUT qualifier, then that name can be up to 39 characters in length. An illegal file name will cause the command to fail.

/PRINT

Prints a listing of the request library definition on the system printer. If you use the /PRINT qualifier without the /OUTPUT qualifier, RDU creates a temporary listing file and deletes it after the print operation is completed. /NOPRINT is the default.

/NOPRINT

Does not print a listing of the request library definition on the system printer. /NOPRINT is the default.

Note

If you use neither the /OUTPUT nor the /PRINT qualifier, RDU writes the request library listing to SYS$OUTPUT but creates no permanent listing file, unless SYS$OUTPUT is defined as a permanent file.

Examples

```
RDU> LIST LIBRARY ACCOUNTING_LIB/OUTPUT=ACCTLIB.LIS
```

Lists the request library definition ACCOUNTING_LIB in the file ACCTLIB.LIS.

```
$ RDU LIST LIBRARY EMPLOYEE_STATUS_LIB/OUTPUT
```

Lists the request library definition EMPLOYEE_STATUS_LIB in the default file EMPLOYEE_STATUS_LIB.LIS.

2.17 LIST REQUEST Command

Provides a listing that shows the source text of the request. You can display the listing on a terminal or direct it to a file or line printer.

Format

LIST REQUEST request-path-name	
Command Qualifiers	*Defaults*
/OUTPUT[=file-spec]	/OUTPUT=SYS$OUTPUT
/[NO]PRINT	/NOPRINT

Prompts

```
RDU>
```

```
$
```

Command Parameter

request-path-name

The CDD path name (given, relative, or full) of the request to be listed.

Command Qualifiers

/OUTPUT[=file-spec]

Lists request information to a file with the specified file name.

If you do not specify a name following the /OUTPUT qualifier, RDU takes the given name of the full request path name up to 39 characters including dollar signs ($) and underscores (_). If that results in a legal file name, then RDU uses that file name with a file type of .LIS. If not, RDU uses the file name RDULIS.LIS.

If you do specify a name following the /OUTPUT qualifier, then that name can be up to 39 characters in length. An illegal file name will cause the command to fail.

/PRINT

Prints a listing of the request on the system printer. If you use the /PRINT qualifier without the /OUTPUT qualifier, RDU creates a temporary listing file and deletes it after the print operation is completed. /NOPRINT is the default.

/NOPRINT

Does not print a listing of the request on the system printer. /NOPRINT is the default.

Note

If you use neither the /OUTPUT nor the /PRINT qualifier, RDU writes the request to SYS$OUTPUT but creates no listing file, unless SYS$OUTPUT is defined as a permanent file.

Examples

```
RDU> LIST REQUEST  ACCOUNTING_REQ/OUTPUT=ACCTREQ.LIS
```

Lists the request ACCOUNTING_REQ in the specified file ACCTREQ.LIS in your default VMS directory.

```
$ RDU LIST REQUEST EMPLOYEE_ADD_REQUEST/OUTPUT
```

Lists the request EMPLOYEE_ADD_REQUEST in the default file EMPLOYEE_ADD_REQUEST.LIS.

2.18 MODIFY LIBRARY Command

Allows you to edit a request library definition that is stored in the CDD.

Format

```
MODIFY LIBRARY request-library-path-name

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 Parameter

request-library-path-name

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

Command Qualifiers

/AUDIT

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

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be

enclosed in parentheses and separated by commas. The audit string can indicate information such as when the request library definition is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/LIST

Creates a listing file in your default or specified VMS directory. The listing file contains the request library definition text and error messages generated by RDU.

If you do not specify a name following the /LIST qualifier, RDU takes the given name of the full request library definition path name up to 39 characters including dollar signs ($) and underscores (_). If that results in a legal file name, then RDU uses that file name with a file type of .LIS. If not, RDU uses the file name RDULIS.LIS.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/NOLIST

Specifies that no listing file be created. /LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LIST=file-spec

Creates a listing file in your default or specified VMS directory with the given file specification. You may use any standard VMS file name up to 39 characters in length. An illegal file name will cause the command to fail.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LOG

Displays a message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

/PRINT

Prints a listing of the request library definition on the system printer. If not used with the /LIST qualifier, /PRINT creates a temporary listing file and deletes it after the print operation is completed. /NOPRINT is the default.

/NOPRINT

Does not print a listing of the request library definition on the system printer. /NOPRINT is the default.

Notes

When you type the MODIFY LIBRARY command, RDU invokes the system-defined logical TDMS$EDIT, which points to the command procedure SYS$COMMON:[SYSEXE]TDMSEDIT.COM by default. The command procedure invokes the editor EDT. If you wish to use a different editor, you can define the logical RDU$EDIT to point to a command procedure file that invokes the text editor of your choice.

When you issue the MODIFY LIBRARY command, RDU:

1. Copies the text of the request library definition into a file
2. Invokes the command procedure defined by RDU$EDIT and displays the text for editing
3. Checks and stores the edited request library definition (if it is error free) when you exit the editor, replacing the old CDD request library definition with the modified one

RDU does not modify a request library definition if:

- The request library definition you specify in the MODIFY LIBRARY command does not already exist in the CDD
- You exit the editor without creating an output file
- RDU finds a syntax error when it is checking the new request library definition text
- RDU is in Validate mode and finds references to requests that are not in the CDD

If RDU cannot modify a request library definition, it displays error messages and does not store the modified request library definition. Instead, it prompts you to check if you want to edit the request library definition again. For example:

```
RDU-E-ERRFNDNOD, error finding node 'ACCOUNT_REQUEST', protocol
'CDD$REQUEST'
-CDD-E-NODNOTFND, Node not found

Do you want to re-try the MODIFY (y or n)?
```

If you enter Y, RDU returns you to the editor. RDU continues to return you to the edit buffer until the request library definition text is correct or you enter N at the prompt.

If RDU is in Novalidate mode, you can modify a request library definition that includes references to requests that are not in the CDD.

Examples

```
RDU>  MODIFY LIBRARY EMPLOYEE_LIBRARY
```

Extracts the text of the request library definition EMPLOYEE_LIBRARY from the CDD and places it in the edit buffer for editing. When you exit the edit buffer, if the EMPLOYEE_LIBRARY is error free, RDU stores it in the CDD.

```
$  RDU MODIFY LIBRARY EMPLOYEE_LIBRARY/PRINT/LOG
```

Extracts the text of the request library definition EMPLOYEE_LIBRARY and places it in your edit buffer for editing. When you exit the edit buffer, RDU checks the text for errors, creates a temporary listing file, and sends a copy of that file to the system printer. If the request library definition is error free, RDU stores it in the CDD and displays a success level message on the screen indicating the request library definition was modified.

2.19 MODIFY REQUEST Command

Allows you to edit a request that is stored in the CDD. If RDU is in Store mode, the MODIFY REQUEST command also creates a request binary structure and stores it in the CDD with the request.

Format

MODIFY REQUEST request-path-name

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

Prompts

```
RDU>
```

```
$
```

Command Parameter

request-path-name

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

Command Qualifiers

/AUDIT

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

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the request is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/LIST

Creates a listing file in your default or specified VMS directory. The listing file contains the request text and error messages generated by RDU.

When used with the /LOG qualifier, the /LIST qualifier creates a listing file that contains information and warning level messages indicating the %ALL mappings that were created and those that were not created and why.

If you do not specify a name following the /LIST qualifier, RDU takes the given name of the full request path name up to 39 characters including dollar signs ($) and underscores (_). If that results in a legal file name, then RDU uses that file name with a file type of .LIS. If not, RDU uses the file name RDULIS.LIS.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/NOLIST

Specifies that no listing file be created. /LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LIST=file-spec

Creates a listing file in your default or specified VMS directory with the given file specification. You may use any standard VMS file name up to 39 characters in length. An illegal file name will cause the command to fail.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LOG

Displays on the terminal screen a message indicating that RDU has successfully completed the operation. If the request is created while in Validate mode, it also displays messages that indicate:

- The request names and the form and record definition names that RDU uses to build the request library file
- The mappings RDU creates for %ALL
- The mapping RDU does not create for %ALL

/NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that RDU has successfully completed the operation. Also does not display information about request names and form and record definition names or mapping messages. /NOLOG is the default.

/PRINT

Prints a listing of the request on the system printer. If the /LIST qualifier is not used, RDU creates a temporary listing file and deletes it after the print operation is completed. /NOPRINT is the default.

/NOPRINT

Does not print a listing of the request on the system printer. /NOPRINT is the default.

/STORE

Stores the request binary structure in the CDD. /STORE is the default in Validate mode.

/NOSTORE

Does not store the request binary structure in the CDD. /STORE is the default in Validate mode.

Notes

When you enter the MODIFY REQUEST command, RDU invokes the system-defined logical TDMS$EDIT, which points to the command procedure SYS$COMMON:[SYSEXE]TDMSEDIT.COM by default. The command procedure

invokes the editor EDT. You can define the logical RDU$EDIT to point to a command procedure file that invokes an editor of your choice.

When you issue the MODIFY REQUEST command, RDU:

1. Copies the text of the request into a file
2. Invokes the command procedure defined by RDU$EDIT and displays the text for editing
3. Checks the text for errors and stores the edited request, if it is error free, when you exit the editor, replacing the old CDD request with the modified one

If RDU is in Validate mode and the /NOSTORE qualifier is not used, the request binary structure is stored in the CDD with the request.

RDU does not modify a request if:

- The request does not already exist in the CDD
- You exit the editor without creating an output file
- RDU finds a syntax error in the modified request text
- RDU is in Validate mode and finds incorrect references to form and record definitions or incorrect mapping instructions

If RDU cannot modify a request, it displays error messages and does not store the modified request. Instead, it prompts you to indicate whether you want to edit the request again. For example:

```
-RDU-E-MISPELKWD, misspelled keyword "DIPLAY" should be "DISPLAY"

Do you want to re-try the MODIFY (y or n)?
```

If you enter Y, RDU returns you to the editor. RDU continues to return you to the editor until the request text is correct or you enter N at the prompt.

If RDU is in Novalidate mode, it will modify a request containing reference or mapping errors.

The default for the /[NO]STORE qualifier is dependent on the Validate mode. If SET VALIDATE is in effect, the default is /STORE. If SET NOVALIDATE is in effect, the default is /NOSTORE. If the request binary structure is stored with

the request, then the BUILD LIBRARY command revalidates the request only if an associated form or record has changed since the request binary structure was created.

Examples

In all of the following examples, the request binary structure is stored with the request in the CDD if RDU is in Store mode.

```
RDU> MODIFY REQUEST EMPLOYEE_REQUEST
```

Copies the text of the request EMPLOYEE_REQUEST into an edit buffer and allows you to modify the request using the features of your editor.

```
$ RDU MODIFY REQUEST EMPLOYEE_REQUEST/PRINT/LOG
```

Copies the text of the request EMPLOYEE_REQUEST into an edit buffer and allows you to change the request using the features of your editor. When you exit the edit buffer, if the request is error free, RDU stores it in the CDD, displays a message indicating the request was modified, and sends a temporary listing file to the system printer.

2.20 REPLACE LIBRARY Command

Replaces a request library definition in the CDD or, if the request library definition does not exist, creates a new request library definition.

Format

REPLACE LIBRARY request-library-path-name [file-spec]

Command Qualifiers	*Defaults*
/[NO]ACL	/ACL
/[NO]AUDIT	/AUDIT
/AUDIT = audit-string	/AUDIT
/[NO]CREATE	/CREATE
/[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 you want to replace.

file-spec

The text file containing the request library definition text that you want to replace. If you do not specify a file type, RDU looks for a file with the default file type .LDF.

If you do not specify a file-spec parameter, RDU expects you to enter the request library definition text from the default file or device defined as SYS$INPUT:

- In Interactive mode, the default is the terminal. You enter the text at the RDUDFN> prompt.
- In Batch mode, the default is the current command stream.

Command Qualifiers

/ACL

Replaces the request library definition with a default CDD access control list. The ACL can grant or deny access to the request library definition based on information contained in the list. For more information about the CDD access control list, refer to the VAX Common Data Dictionary documentation set.

/ACL applies only when the REPLACE LIBRARY command defaults to the CREATE LIBRARY command. /ACL is then the default.

/NOACL

Replaces a request library definition without an access control list. /NOACL applies only when the REPLACE LIBRARY command defaults to the CREATE LIBRARY command. /ACL is the default.

/AUDIT

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

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the request library definition is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/CREATE

Enables RDU to treat the REPLACE LIBRARY command as though it were the CREATE LIBRARY command when it cannot find a request library definition with the specified name in the CDD. /CREATE is the default.

/NOCREATE

Prevents RDU from treating the REPLACE LIBRARY command as though it were the CREATE LIBRARY command. Instead of creating a new request library definition, RDU issues an error message and returns you to the RDU> prompt. /CREATE is the default.

/LIST

Creates a listing in your default or specified VMS directory. The listing file contains the request library definition text and error messages generated by RDU.

If you do not specify a name following the /LIST qualifier, RDU takes the given name of the full request library definition path name up to 39 characters including dollar signs ($) and underscores (_). If that results in a legal file name, then RDU uses that file name with a file type of .LIS. If not, RDU uses the file name RDULIS.LIS.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/NOLIST

Specifies that no listing file be created. /LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LIST=file-spec

Creates a listing file in your default or specified VMS directory with the given file specification. You may use any standard VMS file name up to 39 characters in length. An illegal file name will cause the command to fail.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LOG

Displays a message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that RDU has successfully completed the operation. /NOLOG is the default.

/PRINT

Prints a listing of the request library definition on the system printer. If you do not use the /LIST qualifier with /PRINT, RDU creates a temporary listing file and deletes it after the print operation is completed. /NOPRINT is the default.

/NOPRINT

Does not print a listing of the request library definition on the system printer. /NOPRINT is the default.

Notes

You can replace the request library definition in one of two ways:

- Type the text directly into RDU following the command:

```
REPLACE LIBRARY request-library-path-name
```

- Type the text into a text or command file and pass the file to RDU. You can pass the file to RDU either at DCL level or at the RDU> prompt.

RDU performs the following on a REPLACE LIBRARY command:

1. Finds the request library definition with the name you specified in the command
2. Checks the new request library definition text for errors
3. Deletes the old CDD request library definition and replaces it with the new request library definition

RDU does not replace a request library definition if:

- It finds syntax errors in the request library definition text
- It is in Validate mode and finds errors in the new request library definition's references to requests

If RDU does not replace a request library definition, it issues an error message and returns you to the RDU> prompt (or the DCL prompt if you enter the REPLACE LIBRARY command at DCL level).

In Novalidate mode, RDU replaces the old request library definition with the new one without checking references to requests.

If RDU cannot find a request library definition with the name you specified in the REPLACE LIBRARY command, it treats the REPLACE LIBRARY command as a CREATE LIBRARY command by default and creates a new request library definition. The /ACL and /NOACL qualifiers apply.

You must specify the /NOCREATE qualifier to prevent RDU from creating a new request library definition if there is no existing request library definition.

Examples

```
RDU>   REPLACE LIBRARY EMPLOYEE_LIBRARY EMPLIB.LDF/LOG
```

RDU finds the existing request library definition EMPLOYEE_LIBRARY and replaces it with the library definition text in the file EMPLIB.LDF. When it has successfully replaced the library in the CDD, it displays a success message on the terminal screen.

```
RDU> REPLACE LIBRARY CDD$TOP.PERSONNEL_LIB/PRINT
RDUDFN> REQUEST IS PERSONNEL_REQ;
RDUDFN> REQUEST IS EMPLOYEE_ADD_REQ;
RDUDFN> END DEFINITION;
```

RDU finds the existing CDD request library definition PERSONNEL_LIB and replaces it with the library definition text you enter following the RDUDFN> prompt. RDU creates a temporary listing file, prints it on the system printer, and deletes the temporary file.

```
$  RDU REPLACE LIBRARY INVENTORY_REPORT INVENLIB.LDF
```

Replaces the request library definition INVENTORY_REPORT with the request library definition contained in the file INVENLIB.LDF.

2.21 REPLACE REQUEST Command

Replaces a request in the CDD or, if the request does not exist, creates a new request. If RDU is in Store mode, the REPLACE REQUEST command also creates a request binary structure and stores it in the CDD with the request.

Format

REPLACE REQUEST request-path-name [file-spec]

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

Prompts

```
RDU>
```

```
$
```

Command Parameters

request-path-name

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

file-spec

The text file containing the request text that you want to replace. If you do not specify a file type, RDU looks for the default file type .RDF.

If you do not specify a file-spec parameter, RDU expects you to enter the request text from the default file or device defined as SYS$INPUT:

- In Interactive mode, the default is the terminal. You enter the text at the RDUDFN> prompt.
- In Batch mode, the default is the current command stream.

Command Qualifiers

/ACL

Replaces the request with a default CDD access control list. The ACL can grant or deny access to the request based on information contained in the list. For more information about the CDD access control list, refer to the VAX Common Data Dictionary documentation set.

/ACL applies only when the REPLACE REQUEST command defaults to CREATE REQUEST. /ACL is then the default.

/NOACL

Replaces a request without an access control list. /NOACL applies only when the REPLACE REQUEST command defaults to CREATE REQUEST. /ACL is the default.

/AUDIT

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

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the request is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/CREATE

Enables RDU to treat the REPLACE REQUEST command as though it were the CREATE REQUEST command when it cannot find a request with the specified name in the CDD. /CREATE is the default.

/NOCREATE

Prevents RDU from treating the REPLACE REQUEST command as though it were the CREATE REQUEST command. Instead of creating a new request, RDU issues an error message and returns you to the RDU> prompt. /CREATE is the default.

/LIST

Creates a listing in your default or specified VMS directory. The listing file contains the request text and error messages generated by RDU.

When the /LIST qualifier is used with the /LOG qualifier, the listing file contains information and warning level messages indicating the %ALL mappings that were created and those that were not created and why. (An "A" in the left column of a listing line indicates that the line gives information about a %ALL mapping.)

If you do not specify a name following the /LIST qualifier, RDU takes the given name of the full request path name up to 39 characters including dollar signs ($) and underscores (_). If that results in a legal file name, then RDU uses that file name with a file type of .LIS. If not, RDU uses the file name RDULIS.LIS.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/NOLIST

Specifies that no listing file be created. /LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LIST=file-spec

Creates a listing file in your default or specified VMS directory with the given file specification. You may use any standard VMS file name up to 39 characters in length. An illegal file name will cause the command to fail.

/LIST is the default in Batch mode; /NOLIST is the default in Interactive mode.

/LOG

Displays on the terminal screen a message indicating that RDU has successfully completed the operation. If the request is created while in Validate mode, it also displays messages that indicate:

- The request names and the form and record definition names that RDU uses to build the request library file
- The mappings RDU creates for %ALL
- The mapping RDU does not create for %ALL

/NOLOG is the default.

/NOLOG

Displays no message on the terminal indicating that RDU has successfully completed the operation. Also does not display information about request names and form and record definition names or mapping messages. /NOLOG is the default.

/PRINT

Prints a listing of the request on the system printer. If you do not use the /LIST qualifier with the /PRINT qualifier, RDU creates a temporary listing file and deletes it after the print operation is completed. /NOPRINT is the default.

/NOPRINT

Does not print a listing of the request on the system printer. /NOPRINT is the default.

/STORE

Stores the request binary structure in the CDD. /STORE is the default in Validate mode.

/NOSTORE

Does not store the request binary structure in the CDD. /STORE is the default in Validate mode.

Notes

You can replace the request text in one of two ways:

- Type it directly into RDU following the command:

```
REPLACE REQUEST request-path-name
```

- Type it into a text or command file and pass the file to RDU at either DCL or RDU level.

RDU performs the following on a REPLACE REQUEST command:

1. Finds the request with the name you specified in the command
2. Checks the new text for syntax or mapping errors
3. Deletes the old CDD request and replaces it with the new request

If RDU is in Validate mode and the /NOSTORE qualifier is not used, the request binary structure is stored in the CDD with the request.

RDU does not replace a request if:

- It finds syntax errors in the request text
- It is in Validate mode and finds errors in the new request's references to form or record definitions or mapping instructions

If RDU does not replace a request, it issues an error message and returns you to the RDU> prompt (or the DCL prompt if you type the REPLACE REQUEST at DCL level).

In Novalidate mode, RDU replaces the new request without checking form or record references.

If RDU cannot find a request with the name you specified in the REPLACE REQUEST command, it treats the REPLACE REQUEST command as a CREATE REQUEST command and creates a new request. The /ACL and /NOACL qualifiers apply.

You must specify the /NOCREATE qualifier if you want to prevent RDU from creating a new request if there is no existing request.

The default for the /[NO]STORE qualifier is dependent on the Validate mode. If SET VALIDATE is in effect, the default is /STORE. If SET NOVALIDATE is in effect, the default is /NOSTORE. If the request binary structure is stored with the request, then the BUILD LIBRARY command revalidates the request only if an associated form or record has changed since the request binary structure was created.

Examples

In all of the following examples, the request binary structure is stored with the request in the CDD if RDU is in Store mode.

```
RDU> REPLACE REQUEST EMPLOYEE_REQUEST/LIST/AUDIT
RDUDFN> FORM IS EMPLOYEE_FORM;
RDUDFN> CLEAR SCREEN;
RDUDFN> DISPLAY FORM EMPLOYEE_FORM;
RDUDFN> END DEFINITION;
```

RDU finds the request EMPLOYEE_REQUEST in the CDD and replaces it with the request following the REPLACE REQUEST command. RDU also creates a listing file and places the standard audit string with the replaced request text.

```
$ RDU REPLACE REQUEST PERSONNEL_REQUEST PERSON.RDF/PRINT
```

RDU finds the PERSONNEL_REQUEST in the CDD and replaces the request with the text in the file PERSON.RDF. RDU also sends a listing of the file to the system printer.

2.22 SAVE Command

Saves in a file the most recent command you entered in RDU and its associated text.

Format

```
SAVE file-spec
```

Prompt

```
RDU>
```

Command Parameter

file-spec

The name of the file in which you want to save the last command and the associated text. You can use any standard VMS file name up to 39 characters in length. If you do not specify a directory, RDU puts it in the directory in which you are currently working. The default file type is .SAV.

Note

You can use this command to save a request or library text you create and want to place in a file for printing or editing.

Example

```
RDU> CREATE REQUEST EMPLOYEE_TEST
RDUDFN> FORM IS TEST1;
RDUDFN> RECORD IS TEST2;
RDUDFN> DISPLAY FORM TEST1;
RDUDFN> END DEFINITION;
RDU> SAVE
Save to file      : EMPLOYEE.EDT
%RDU-I-SAVETOFIL, previous command SAVEd to file DUA2:[SMITH.WORK]EMPLOYEE.EDT;1
RDU> EXIT

$  TY EMPLOYEE.EDT
CREATE REQUEST EMPLOYEE_TEST
FORM IS TEST1;
RECORD IS TEST2;
DISPLAY FORM TEST1;
END DEFINITION;
```

RDU prompts you for a file name if you do not specify it following the SAVE command. It writes the CREATE REQUEST command and the text following the command to the file you specify, EMPLOYEE.EDT in the directory [SMITH.WORK]. The example shows the file contents that RDU saves.

2.23 SET DEFAULT Command

Sets RDU to point to a CDD directory for as long as you are in RDU or until you set RDU to point to a new directory.

Format

```
SET DEFAULT CDD-directory-path-name
```

Prompt

```
RDU>
```

Command Parameter

CDD-directory-path-name

The CDD path name (given, relative, or full) of the default CDD directory.

Notes

Once you specify a default CDD directory, all path names are interpreted as starting at that level. You can override this default directory setting by specifying the full path name beginning with CDD$TOP.

The CDD default directory setting ends when you exit RDU. You must reset the CDD default directory when you reenter RDU.

If you want to create a CDD directory setting that exists every time you enter RDU or until you reset it, you can either:

- Enter the SET DEFAULT command in your startup command file RDUINI-.COM. (See the @file-spec command.)
- Use the DCL DEFINE command to define the logical name CDD$DEFAULT in your login command file. (See Chapter 2 of the *VAX TDMS Request and Programming Manual* for more information.)

Examples

```
RDU>  SET DEFAULT CDD$TOP.PERSONNEL.ACCOUNTING.SMITH
```

Sets the CDD default directory to CDD$TOP.PERSONNEL.ACCOUNTING.SMITH.

```
$ EDIT RDUINI.COM
SET DEFAULT CDD$TOP.YOUR_HOME_DIRECTORY
CTRL/Z
   *EXIT
$ RDU
RDU>  SHOW DEFAULT
current CDD default path is 'CDD$TOP.YOUR_HOME_DIRECTORY'
```

Places the SET DEFAULT command in the startup command file that is executed each time you invoke RDU.

2.24 SET [NO]LOG Command

Creates a log and writes further output to a default or specified log file.

Format

```
SET [NO]LOG [file-spec]
```

Prompt

```
RDU>
```

Command Parameter

file-spec

The name of the file to which you want RDU to log. Use the standard VMS file specification.

If you specify the name previously specified in the current RDU session, RDU creates a new version of that log file. If you specify a different name, RDU closes the first log file and opens the new one.

(If you do not specify a file, RDU logs to a default file specification as described in the following Notes section.)

Notes

RDU logs the following data in the log file:

- An exact copy of each of the commands you enter or a copy of the commands executed by RDU from an indirect command file (if the indirect command file is executed in Set Verify mode)
- All request or request library instructions that follow those commands
- RDU messages preceded by an exclamation point (!) comment character

You can issue the SET LOG command without specifying a file name. RDU creates a default file in one of three ways:

- RDU creates a default log file by translating the logical name RDULOG.
- If the system does not have an equivalent name for RDULOG, RDU uses the file name RDULOG.LOG.
- If a current log file already exists when you issue the SET LOG command without naming a file, RDU issues a warning and continues to use the current log file.

The SET LOG command remains in effect for the entire session. If you exit RDU and reenter, you must issue the SET LOG command again. RDU opens a new log file.

The SET NOLOG command deactivates logging in RDU. SET NOLOG is the default. If you want SET LOG to be the default, you can include a SET LOG command in your RDUINI.COM startup command file.

Examples

```
RDU> SET LOG
```

RDU automatically logs to the RDULOG.LOG file or to the logical translation of RDULOG.

```
RDU> SET LOG EMPLREQ.LOG
```

RDU logs to the EMPLREQ.LOG file.

2.25 SET [NO]VALIDATE Command

Places RDU in Validate or Novalidate mode.

Format

```
SET [NO]VALIDATE
```

Prompt

```
RDU>
```

Notes

Validate Mode:

By default, RDU is in Validate mode. It checks requests and request library definitions for valid references to form definitions, record definitions, and requests.

If you create, replace, or modify a *request* in Validate mode, RDU checks that:

- The form names and record names you refer to are in the CDD
- The form field names you specify in the request match the form field names in the CDD form definition
- The record field names you specify in the request match the record field names in the CDD record definitions
- In a %ALL mapping, at least one form field on the active form has an identically named record field
- The mappings you define in the request are valid mappings:
  - The data types and lengths of fields you map to each other are compatible.
  - The types of the form and record fields you map are compatible.

If a request is valid, RDU allows you to create, modify, or replace a request in the CDD. Otherwise, if a single explicit mapping instruction is not valid, RDU issues error or warning level messages indicating it cannot validate the request.

If the request contains only %ALL mappings, RDU creates only those mappings that are valid and stores the request in the CDD.

If you create, modify, or replace a *request library definition* in Validate mode, RDU checks that the requests you specify are in the CDD.

If they are in the CDD, RDU allows you to create, modify, or replace a request library definition. Otherwise RDU displays an error level message indicating why it cannot create, modify, or replace the request library definition.

You can create, modify, and replace requests or library definitions in Validate mode only if they contain correct references and mappings.

In Validate mode, if you use the /STORE qualifier with a CREATE REQUEST, MODIFY REQUEST, REPLACE REQUEST, or VALIDATE REQUEST command, RDU will store the corresponding request binary structure in the CDD. If you use the /STORE qualifier in Validate mode with a VALIDATE LIBRARY command, RDU will store the request binary structure for each request contained in the request library definition.

Once the request binary structures are stored in the CDD, RDU will revalidate the requests during execution of a BUILD LIBRARY command only when any related forms or records have changed. This process decreases the amount of time needed to build a request library file.

/STORE is the default for Validate mode.

Novalidate Mode:

If RDU is in Novalidate mode, you can create, modify, or replace requests and request library definitions in the CDD, and RDU does not check for correct mappings or references to forms, records, or requests.

After you issue a SET NOVALIDATE command, RDU remains in Novalidate mode until the SET VALIDATE command is issued or you exit RDU.

In Novalidate mode, if you use the /STORE qualifier with a CREATE REQUEST, MODIFY REQUEST, REPLACE REQUEST, or VALIDATE REQUEST command, RDU will signal an error and will not perform the appropriate operation.

If you use the /NOSTORE qualifier when RDU is in Validate mode, RDU will not store any request binary structures in the CDD. The size of the CDD will decrease, but the time needed to build a request library file will increase.

You should use the /NOSTORE qualifier only when RDU is in Novalidate mode or CDD space is limited.

/NOSTORE is the default in Novalidate mode.

Examples

```
RDU> SET VALIDATE
```

Places RDU in the Validate mode. RDU checks all requests or request library definitions you create, modify, or replace in this mode for valid references and mapping instructions.

```
RDU> SET NOVALIDATE
```

Places RDU in the Novalidate mode. RDU does not check all requests or request library definitions you create in this mode for valid references and mapping instructions. If a request does contain incorrect mappings or references, you cannot build a request library file until you correct those errors.

2.26 SET [NO]VERIFY Command

Enables the display (on a terminal) or printing of the contents of RDU command procedures.

Format

```
SET [NO]VERIFY
```

Prompt

```
RDU>
```

Notes

By default, RDU is set to:

- Noverify in Interactive mode. When RDU executes commands from an indirect command file, it does not display those commands to SYS$OUTPUT.
- Verify in Batch mode. In Batch mode, RDU writes the command it is executing to the batch log file.

The RDU default can be changed by a SET [NO]VERIFY command in either a command procedure you explicitly invoke or in your RDUINI.COM file.

Examples

```
RDU> SET VERIFY
RDU> @EMPREQ
```

Places RDU in Verify mode. When you pass the command file EMPREQ.COM to RDU, it displays the commands it executes from that file.

```
RDU> SET NOVERIFY
RDU> @ACCTREQ
```

Places RDU in Noverify mode. When you pass the command file ACCTREQ.COM to RDU, it does not display the commands it executes from that file.

2.27 SHOW DEFAULT Command

Displays the current CDD default directory.

Format

```
SHOW DEFAULT
```

Prompts

```
RDU>
```

```
$
```

Note

Shows the current CDD default directory. You can define the default in RDU by using the SET DEFAULT command or at DCL level by using the DEFINE command to set CDD$DEFAULT to the desired CDD directory.

Example

```
RDU> SHOW DEFAULT
current CDD default path is 'CDD$TOP.TEST'
```

Indicates that the current default directory setting is CDD$TOP.TEST. When you process (create, list, and so on) requests or request library definitions using the given name or a relative path name, RDU will look for or store the definitions in CDD$TOP.TEST.

2.28 SHOW LOG Command

Displays information about the current logging status of RDU.

Format

```
SHOW LOG
```

Prompt

```
RDU>
```

Note

The SHOW LOG command tells whether logging is taking place, and, if so, the name of the file to which the log is written. If logging is not taking place, RDU indicates that logging is not taking place to RDULOG.

Example

```
RDU> SHOW LOG
not logging to file RDULOG
```

RDU indicates it is not logging to a default file.

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.

2.30 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.

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the request library definition is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/LOG

Displays on the terminal screen messages that indicate:

- The request names and the form and record definition names that are referenced in the request library definition
- The mappings that are correct for all the %ALL mapping instructions in the request
- The mappings that are incorrect for all the %ALL mapping instructions in the request
- Whether or not the operation successfully completed

RDU does not generate messages about the individual explicit mappings that are correct. /NOLOG is the default.

/NOLOG

Displays no messages on the terminal indicating that RDU has successfully completed the operation. Also does not display information about request names and form and record definition names or mapping messages. /NOLOG is the default.

/STORE

Creates a request binary structure for each of the requests that the request library refers to and stores these request binary structures in the CDD. /STORE is the default in Validate mode.

/NOSTORE

Does not store the request binary structures in the CDD. /STORE is the default in Validate mode.

Notes

You can use this command to determine whether a request library definition that you have created is valid, that is, whether the requests named in the library definition:

- Exist in the CDD
- Contain correct references to CDD forms and records and specify legal mappings (are valid requests)

If the requests do not exist or if they contain reference or mapping errors (are invalid requests), RDU displays error messages. RDU indicates which, if any, requests are not valid. See VALIDATE REQUEST for further explanation about valid requests.

You can use this command to determine if a request library is valid and points to valid requests without having to build a request library file.

You can issue this command when RDU is in either Novalidate or Validate mode. In either case, RDU performs full validation.

Validation of a request proceeds in three phases:

1. If the request was validated previously, RDU checks that any related forms or records have not changed. If so, the request is valid.
2. If the request has not been validated previously, or if any related forms or records have changed, RDU validates the request.
3. RDU stores the request binary structure in the CDD. This third phase occurs only if:
   - The second phase was necessary
   - The validation process completed successfully
   - RDU is in Store mode

When you use the VALIDATE LIBRARY command, RDU is in Validate mode and the default is /STORE. If the request binary structure is stored with the request, then the BUILD LIBRARY command revalidates the request only if an associated form or record has changed since the request binary structure was created.

Examples

```
RDU> VALIDATE LIBRARY EMPLOYEE_LIBRARY
```

RDU checks EMPLOYEE_LIBRARY in your default CDD directory for correct references to requests, forms, and records and for correct mapping instructions. If RDU is in Store mode, the VALIDATE LIBRARY command also creates a request binary structure for each request in the request library definition.

```
$ RDU VALIDATE LIBRARY CDD$TOP.SAMPLE.EMPLOYEE_LIBRARY
```

RDU checks EMPLOYEE_LIBRARY in the CDD directory CDD$TOP.SAMPLE for correct references to requests, forms, and records and for correct mapping instructions. If RDU is in Store mode, the VALIDATE LIBRARY command also creates a request binary structure for each request in the request library definition.

2.31 VALIDATE REQUEST Command

Checks a request for correct references to form and record definitions and for legal mappings. If RDU is in Store mode, the VALIDATE REQUEST command also creates a request binary structure and stores it in the CDD with the request.

Format

VALIDATE REQUEST request-path-name

Command Qualifiers	*Defaults*
/[NO]AUDIT	/AUDIT
/AUDIT = audit-string	/AUDIT
/[NO]LOG	/NOLOG
/[NO]STORE	/STORE

Prompts

```
RDU>
```

```
$
```

Command Parameter

request-path-name

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

Command Qualifiers

/AUDIT

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

The optional audit string consists of one or more single words, quoted strings, text from a file, or a combination of these three items. A list of items must be enclosed in parentheses and separated by commas. The audit string can indicate information such as when the request is created, accessed, or changed.

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

For more information, see the beginning of this chapter.

/LOG

Displays on the terminal screen messages that indicate:

- The request names and the form and record definition names that are referenced in the request library definition
- The mappings that are correct for all the %ALL mapping instructions in the request
- The mappings that are incorrect for all the %ALL mapping instructions in the request
- Whether or not the request is valid

(RDU does not generate messages about the individual explicit mappings that are correct.) /NOLOG is the default.

/NOLOG

Displays no messages on the terminal indicating that RDU has successfully completed the operation. Also does not display information about request names and form and record definition names or mapping messages. /NOLOG is the default.

/STORE

Stores the request binary structure in the CDD. /STORE is the default in Validate mode.

/NOSTORE

Does not store the request binary structure in the CDD. /STORE is the default in Validate mode.

Notes

You can use this command to determine if a request you create is valid. The VALIDATE REQUEST command verifies that:

- The form names and record names you refer to are in the CDD
- The form field names you specify in a request match the form field names in the CDD form definition
- The record field names you specify in a request match the record field names and structures in the CDD record definition
- The mappings you define in the request are correct:
  - The data types and lengths of fields you map to each other are compatible.
  - The types of the form and record fields you map are compatible.

You use this command if you wish to validate requests before you issue a BUILD command.

You can issue this command when RDU is in either Validate or Novalidate mode. In either case, RDU performs full validation.

Validation of a request proceeds in three phases:

1. If the request was validated previously, RDU checks that the related forms and records have not changed. If they have not, the request is valid.
2. If the request has not been validated previously, or if any related forms or records have changed, RDU validates the request.
3. RDU stores the request binary structure in the CDD. This third phase occurs only if:
   - The second phase was necessary
   - The validation process completed successfully
   - RDU is in Store mode

When you use the VALIDATE REQUEST command, RDU is in Validate mode and the default is /STORE. If the request binary structure is stored with the request, then the BUILD LIBRARY command revalidates the request only if an associated form or record has changed since the request binary structure was created.

Examples

```
RDU> VALIDATE REQUEST EMPLOYEE_REQUEST
```

RDU checks that EMPLOYEE_REQUEST in your default CDD directory contains correct references to forms and records and correct mapping instructions. If RDU is in Store mode, the VALIDATE REQUEST command also creates a request binary structure for the request.

```
$ RDU VALIDATE REQUEST CDD$TOP.ACCOUNTING.EMPLOYEE_REQUEST
```

Checks that EMPLOYEE_REQUEST in the CDD directory CDD$TOP.ACCOUNTING contains correct references to forms and records and correct mapping instructions. If RDU is in Store mode, the VALIDATE REQUEST command also creates a request binary structure for the request.

Request and Request Library Instructions 3

This chapter provides complete information on the syntax and use of all the request and request library definition instructions. The instructions are in alphabetical order.

Each section contains the following categories, as applicable:

Format	Provides the syntax for the instruction.
Prompts	Shows the prompts for each command.
Instruction Parameters	Explains each parameter.
Instruction Modifiers	Explains each modifier and how to use it.
Notes	Provides information about using the instruction.
Examples	Provides examples of the instruction.

Note that instruction keywords (required words) should not be used in requests or libraries for names of requests, forms, records, request library definitions, record or field names, or any other variable names.

3.1 [NO] BLINK FIELD Instruction

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

Format

```
[NO] BLINK 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 BLINK or NO BLINK instruction in a request, it overrides:

- A Blink or No Blink attribute assigned in a form definition.
- A BLINK or NO BLINK 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 BLINK or NO BLINK instruction used within a conditional instruction supersedes one in a base request or any outer conditional instruction.

Only the Blink or No Blink attribute of the form field is affected. All other video attributes of the field and the content of the field remain unchanged. For instance, if you specify a BOLD instruction in addition to a BLINK instruction, the field is both bolded and blinking.

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.

Only the Bold or No Bold attribute of the form field is affected. All other video attributes of the field and the content of the field remain unchanged. For instance, if you specify a BLINK instruction in addition to a BOLD instruction, the field will be both bolded and blinking.

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.

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.

3.4 CONTROL FIELD IS Instruction

Specifies that request instructions be executed only if specific conditions exist in an application program.

Format

```
CONTROL FIELD IS record-field

      quoted-string :   match-instruction;
                      [ ...                ]

    ...

   [ ANYMATCH :   match-instruction;    ]
   [              [ ...                ] ]

   [  NOMATCH :   match-instruction;    ]
   [              [ ...                ] ]

END CONTROL FIELD;
```

Prompt

```
RDUDFN>
```

Instruction Parameters

record-field

A record field in the CDD record definition used by the request. The record field you specify is called the *control value*. It can be a simple field or an array field and must have a data type of TEXT.

If the control value is an array (with a range of contiguous subscript values), the set of subscript values is called a dependent range. The dependent range determines the values of two variable subscripts (dependent names): %LINE and %ENTRY. (See Chapter 9 in the *VAX TDMS Request and Programming Manual*, Using an Array as a Control Value, for more information.)

Note that the words "CONTROL FIELD IS record-field" are not followed by a semicolon.

quoted-string

Any text string enclosed in double or single quotation marks. TDMS tests the quoted string value against the control value. If the values match, TDMS executes the match instructions following the quoted string. The quoted string must be followed by a colon. The string must be no longer than the matching control field and it cannot exceed a single line.

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."

TDMS does not distinguish between uppercase and lowercase letters in case values. For example, TDMS treats the following as equal: "yes", "YeS", "YES".

ANYMATCH:

A keyword. TDMS executes the match instructions following the ANYMATCH keyword if the value in a control value matches *any* quoted string case value associated with that CONTROL FIELD IS instruction.

You must use a colon (:) after the ANYMATCH keyword.

NOMATCH:

A keyword. TDMS executes the match instructions following the NOMATCH keyword if the value in a control value does not match *any* quoted string case value associated with that CONTROL FIELD IS instruction.

You must use a colon (:) after the NOMATCH keyword.

match-instruction

Any legal request instruction except request header instructions:

- FORM IS
- RECORD IS

Notes

When an application program calls a request containing a conditional instruction (CONTROL FIELD IS), TDMS evaluates the case values you specify. If any of these case values match the value in the control value, TDMS executes the match instructions following the case value.

The value in the control value is typically placed there by the action of the program. However, you can design a request that maps values to the control value, either directly from the operator or from literals within the request.

You can specify any number of CONTROL FIELD IS instructions in a single request. However, if you do use more than one CONTROL FIELD IS instruction, you must make sure there are no conflicting instructions. For example, the following conditional request would give unpredictable results: two literal messages are mapped to a single form field depending on previous operator input. In this example, only one of the messages is sent to the form, and the other message is ignored.

```
CONTROL FIELD IS CONTROL_EMPLOYEE_GENDER

    "M":  OUTPUT "MALE" TO GENDER_FRM_FIELD;
    "F":  OUTPUT "FEMALE" TO GENDER_FRM_FIELD;

END CONTROL FIELD;

CONTROL FIELD IS CONTROL_EMPLOYEE_DEPENDENT

    "Y":  OUTPUT "YES" TO GENDER_FRM_FIELD;
    "N":  OUTPUT "NO" TO GENDER_FRM_FIELD;

END CONTROL FIELD;
```

You can also *nest* conditional instructions; that is, you can specify an inner CONTROL FIELD IS instruction following a case value. If you do nest conditional instructions, TDMS executes the inner conditional instruction only if the outer case value and control field value match.

TDMS executes the request instructions within conditional requests in the following order:

1. Output instructions in the base request
2. Output instructions in the outer control field
3. Output instructions in the nested control field

After executing the output instructions, TDMS executes the input instructions in the same order, proceeding from the base request to the innermost nested control field.

You can nest conditional instructions as many times as you want (unless your control field is an array). As with multiple control fields, you must be careful to avoid conflicting nested mapping instructions. Because TDMS executes nested instructions last, if there is a run-time conflict, the inner instructions supersede. At run time, then, in the case of conflicting mappings, the operator sees only the innermost output or input mappings.

If the control field is an array with a dependent range (rather than a single subscript or a series of single subscript values) you cannot nest another array control field. See Chapter 9 in the *VAX TDMS Request and Programming Manual* on Using an Array As a Control Value, for more information on dependent ranges.

You can use conditional instructions to:

- Reduce the number of requests you write for a single application
- Remove conditional expressions such as IF-THEN-ELSE or SELECT-CASE OF, and the resulting statement execution, from program code and place them within a request
- Simplify maintaining and developing an application by placing all the terminal I/O instructions and logic within a single request
- Provide a convenient way to validate data from each form field and display error messages relating to operator input

Examples

```
RDUDFN> FORM IS      CDD$TOP.EMPLOYEE_INITIAL_FORM;
RDUDFN> FORM IS      CDD$TOP.EMPLOYEE_MENU_FORM;
RDUDFN> RECORD IS    CDD$TOP.EMPLOYEE_RECORD;
RDUDFN> RECORD IS    CDD$TOP.EMPLOYEE_COND_WORKSPACE;
RDUDFN>
RDUDFN>      CLEAR SCREEN;
RDUDFN>
RDUDFN>      CONTROL FIELD IS START_PROGRAM
RDUDFN>
RDUDFN>          "FIRST":
RDUDFN>              DISPLAY FORM EMPLOYEE_INITIAL_FORM;
RDUDFN>              RETURN "      " TO START_PROGRAM;
RDUDFN>              WAIT;
RDUDFN>
RDUDFN>          NOMATCH:
RDUDFN>              DISPLAY FORM EMPLOYEE_MENU_FORM;
RDUDFN>              INPUT SELECTION TO SELECTION;
RDUDFN>              INPUT EMPLOYEE_NUMBER TO EMPLOYEE_NUMBER;
RDUDFN>
RDUDFN>       END CONTROL FIELD;
RDUDFN> END DEFINITION;
```

When the application program calls this request the first time, because the string "FIRST" is in the control value, TDMS executes the instructions following the case value "FIRST". The RETURN TO instruction clears the control value START_PROGRAM. The second time the program calls the request, because the control value no longer matches "FIRST", TDMS executes the instructions following the NOMATCH case value. It displays the employee menu form and collects a menu selection and employee number from the operator.

```
RDUDFN> CONTROL FIELD IS JOB_CODE
RDUDFN>     "H": OUTPUT "Hourly" TO WAGE_FIELD;
RDUDFN>     "S": OUTPUT "Salaried" TO WAGE_FIELD;
RDUDFN>
RDUDFN>         CONTROL FIELD IS BENEFIT_CODE
RDUDFN>             "S": OUTPUT "Participating in stock option" TO BENEFITS_FIELD;
RDUDFN>             NOMATCH: OUTPUT "No benefits" TO BENEFITS_FIELD;
RDUDFN>         END CONTROL FIELD;
RDUDFN>
RDUDFN>  END CONTROL FIELD;
```

When the application program calls this request, TDMS executes the outer output instruction first, displaying either "Hourly" or "Salaried" in the form field WAGE_FIELD. It executes the nested output instruction second, displaying either "Participating in stock option" or "No benefits" in the form field BENEFITS_FIELD. Note that you can specify either lowercase or uppercase characters as case values.

3.5 [NO] DEFAULT FIELD Instruction

Displays the default field contents specified in the form definition.

Format

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

Prompt

```
RDUDFN>
```

Instruction Parameters

form-field

The name of the form field you want to be displayed with default values. The field must be on the active form. You can specify a single form field or a list of form fields separated by commas.

%ALL

All the fields on the active form.

Notes

Only the contents of the named fields are reset, not the video attributes.

You may want to use this instruction in conjunction with the USE FORM instruction to display the form with previously entered field contents except for particular fields.

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

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.

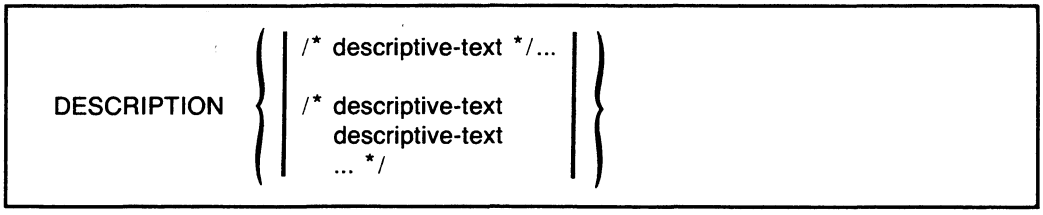
```
RDUDFN> NO DEFAULT FIELD EMPLOYEE, BADGE, DEPT;
```

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

3.6 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.

Examples

```
RDUDFN> FORM IS EMPLOYEE_FORM;
RDUDFN> DESCRIPTION  /* This form collects all
RDUDFN> operator input data for EMPLOYEE_REC.  The
RDUDFN> badge number of an employee is displayed, and
RDUDFN> all other form fields (NAME, ADDRESS, DEPARTMENT)
RDUDFN> are input fields.*/;
```

The descriptive text on the form is stored after the FORM IS instruction. The DESCRIPTION instruction cannot be embedded in another instruction.

```
RDUDFN> DESCRIPTION
RDUDFN> /* This request library definition names all
RDUDFN> the requests used in the Employee application */;
```

The descriptive text can span several lines.

3.7 DISPLAY FORM Instruction

Displays the specified form with its form-defined defaults. Sets the active form for the request.

Format

```
DISPLAY FORM form-name [WITH OFFSET offset-value];
```

Prompt

```
RDUDFN>
```

Instruction Parameter

form-name

The form name can be either:

- The given name of an existing form definition.
- The unique form name specified in the WITH NAME clause of the FORM IS instruction. If the WITH NAME clause is specified in the FORM IS instruction, you *must* use this parameter.

Instruction Modifier

WITH OFFSET offset-value

The keywords WITH OFFSET and a number that specifies a beginning line at which the form is displayed on the screen. The offset value is added to the number of the line where the form would be displayed without the WITH OFFSET modifier. The offset value is a number between 0 and +22. The plus sign preceding the number is optional.

When you validate the request or build the request library file, if the offset value you specify would cause any portion of the form to be displayed past the 23rd line of the screen, RDU issues an error level message and does not build the request library file.

Notes

The form you specify in the DISPLAY FORM instruction is the active form; that is, it is the only form to and from which data can be mapped during that call to the request.

If the form you specify already appears on the screen from a previous request call, TDMS discards all the data on the form and resets the field contents and field video to the form-defined defaults.

If you specify output mappings or video changes with a DISPLAY FORM instruction, the output instructions override the form-defined defaults.

Examples

```
RDUDFN>  DISPLAY FORM EMPLOYEE_STATUS_FORM WITH OFFSET 2;
```

Displays the form EMPLOYEE_STATUS_FORM beginning on line 3 of the screen. That is, WITH OFFSET 2 causes the form to be displayed two lines below where it would be displayed if no WITH OFFSET modifier were used.

```
RDUDFN>  FORM IS EMPLOYEE_STATUS_FORM WITH NAME EMP_FORM;
RDUDFN>  DISPLAY FORM EMP_FORM;
```

Displays the form EMPLOYEE_STATUS_FORM, using the unique name EMP_FORM specified in the FORM IS instruction.

3.8 END DEFINITION Instruction

Indicates the end of a request or a request library definition.

Format

```
END DEFINITION;
```

Prompt

```
RDUDFN>
```

Notes

The END DEFINITION instruction must be the last instruction in every request and request library definition. It must always be followed by a semicolon.

In a request, the END DEFINITION instruction causes RDU to stop accepting request instructions and checking for syntax errors. In Validate mode, RDU then begins checking for mapping errors. In Novalidate mode, RDU does not check for mapping errors.

When the error checking is done, RDU returns you to the RDU> prompt (or to the DCL prompt if you are at DCL level).

In a request library definition, the END DEFINITION instruction causes RDU to stop accepting request library definition instructions and checking for syntax errors.

When the error checking is done, RDU returns you to the RDU> prompt (or to the DCL prompt if you are at DCL level).

Example

```
RDUDFN> END DEFINITION;
RDU>
```

Ends the current request definition (assuming this is a request). If there are no syntactic errors in the request source text and the mappings are valid (if RDU is in Validate mode), RDU stores the request in the CDD and returns to the RDU> prompt.

3.9 FILE IS Instruction

Identifies the name of a request library file that RDU creates if you issue the BUILD LIBRARY command.

Format

```
FILE IS "file-spec";
```

Prompt

```
RDUDFN>
```

Instruction Parameter

"file-spec"

The standard VMS file specification format. It must be enclosed in matched double or single quotation marks. You can include the full file specification using the format:

"node::device:[directory]filename.type;version"

You can use defaults for any element of the file specification other than the file name. The default file type is .RLB.

The file specification can also be a logical name that RDU translates when the request library file is built.

Notes

You can use no more than one FILE IS instruction in a request library definition.

If you do not use the FILE IS instruction, RDU looks for a request library file specification in the BUILD command when it builds a request library file.

If you do not specify a file in either the request library definition with a FILE IS instruction or in the BUILD command, RDU issues an error message and does not build a request library file.

Use this instruction only when creating, modifying, or replacing a request library definition in the CDD.

Examples

```
RDUDFN> FILE IS "DBA1:[SMITH]PERSONNEL.RLB";
```

Specifies that RDU name a request library file PERSONNEL.RLB on disk DBA1, directory SMITH.

```
RDUDFN> FILE IS 'DEPARMT.RLB';
```

Specifies that RDU name a request library file DEPARMT.RLB in your VMS default directory.

3.10 FORM IS Instruction

Identifies a form or forms to be used in a request or included in a request library definition.

Format

```
{ FORM IS  }
{ FORMS ARE } form-path-name [WITH NAME unique-form-name],...;
```

Prompt

```
RDUDFN>
```

Instruction Parameter

form-path-name

The CDD path name (given, relative, or full) of an existing form 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 forms, or a record and a form, cannot use the same name within the body of a request or a request library definition. If two form given names, or a form given name and a record 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 or request library definition.

Instruction Modifier

WITH NAME unique-form-name

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

The unique form name, if specified, is the one you use in subsequent DISPLAY FORM and USE FORM instructions within the body of a request. If the unique name is not specified, use the given name.

Notes

The form named in the FORM IS instruction must exist in the CDD if you are running RDU in Validate mode (the default mode). If RDU is not in Validate mode, it does not check for the existence of the specified form definition.

The FORM IS instruction can be used in both a request and a request library definition.

If the FORM IS instruction is used in a request, it must appear in the header portion of a request. You cannot define mappings to a form unless you identify that form in a FORM IS instruction and either a DISPLAY FORM or a USE FORM instruction.

You can specify many FORM IS instructions in a single request. However, you cannot specify Help forms, and you must specify only forms that are used in the DISPLAY FORM or USE FORM instructions within a request.

There is no restriction on the number of FORM IS instructions in a request library definition.

TDMS, however, can map data to and from, or manipulate the contents or video attributes of, only one form during a single call to a request. That is, only one form is active during a single request call. You indicate which form is active by using the DISPLAY FORM or USE FORM instruction.

For example, a request can contain the following instructions:

```
FORMS ARE EMPLOYEE_FORM,
          ACCOUNTING_FORM,
          PERSONNEL_FORM;

RECORD IS EMPLOYEE_RECORD;

CONTROL FIELD IS  SELECT_FORM_CODE

   "EMPL": USE FORM EMPLOYEE_FORM;
           INPUT EMPLOYEE_BADGE TO EMPLOYEE_BADGE;

   "ACCT": USE FORM ACCOUNTING_FORM;
           INPUT SALARY_CLASS TO SALARY_CLASS;

   "PERS": USE FORM PERSONNEL_FORM;
           INPUT DEPARTMENT_NO TO DEPARTMENT_NO;

END CONTROL FIELD;

END DEFINITION;
```

When an application program calls this request, TDMS selects only one of the forms to map data to and from, depending on the value in the control value.

To use TDMS forms with DATATRIEVE, you include a FORM IS instruction in a request library definition. For more information, see the chapter on using forms with DATATRIEVE in the *VAX TDMS Forms Manual*.

Examples

```
RDUDFN> FORM IS EMPLOYEE_FORM;
```

Specifies a form, EMPLOYEE_FORM, using the given name.

```
RDUDFN> FORM IS CDD$TOP.ACCOUNTING.PAYROLL.EMPLOYEE_FORM;
```

Specifies a form, CDD$TOP.ACCOUNTING.PAYROLL.EMPLOYEE_FORM, using the full path name.

```
RDUDFN> FORM IS CDD$TOP.ACCOUNTING.EMPLOYEE WITH NAME ACCOUNT_FORM;
RDUDFN> FORM IS CDD$TOP.FINANCE.EMPLOYEE WITH NAME  FINANCE_FORM;
```

Specifies two forms, CDD$TOP.ACCOUNTING.EMPLOYEE and CDD$TOP.FINANCE.EMPLOYEE, using the full path names and the WITH NAME modifier. In subsequent references to these forms within this request, you *must* use the unique names ACCOUNT_FORM and FINANCE_FORM.

```
RDUDFN> FORMS ARE EMPLOYEE_FORM, ACCOUNTING_FORM, PERSONNEL_FORM;
```

Specifies several forms using the given names.

3.11 %INCLUDE Instruction

Extracts all the text from the specified file and includes it in the current request or request library definition.

Format

```
%INCLUDE "file-spec"[;]
```

Prompt

```
RDUDFN>
```

Instruction Parameter

"file-spec"

The standard VMS file specification format. It must be enclosed in matched double or single quotation marks. You can include the full file specification using the format:

"node::device:[directory]filename.type;version"

You can use defaults for any element of the file specification other than file name. The default file type is .COM.

The file specification can also be a logical name that RDU translates before it extracts the text from the file.

Notes

You can issue the %INCLUDE instruction:

- After you issue an RDU command, such as CREATE REQUEST, and want to include a file containing source text
- From within a text file that you pass to RDU

The need for a semicolon at the end of the instruction is based on the contents of the %INCLUDE file. If the file ends with a semicolon, you do not need a semicolon after %INCLUDE. Otherwise, you must use a semicolon after %INCLUDE.

The text in the %INCLUDE file must have the same syntax as if it were entered directly in RDU. After RDU includes the text from the %INCLUDE file, it returns to the RDUDFN> prompt (or to the RDU> prompt if an END DEFINITION instruction is in the %INCLUDE file).

The %INCLUDE instruction can be nested to any depth.

The text of a %INCLUDE file is not stored in the CDD. If you process a request or request library definition stored in the CDD that contains a %INCLUDE instruction, RDU looks for the %INCLUDE file in the VMS directory specified in the %INCLUDE instruction.

The text of the %INCLUDE file is:

- Displayed to the output file or device defined as SYS$OUTPUT, if SET VERIFY is in effect
- Placed in the log file preceded by the !%INC> prompt, if SET LOG and SET VERIFY are in effect

Examples

```
RDU> CREATE REQUEST EMPLOYEE_REQUEST
RDUDFN> FORM IS EMP_FORM;
RDUDFN> RECORD IS EMP_REC;
RDUDFN> %INCLUDE "EMPLOYEE.DAT";
RDUDFN> END DEFINITION;
```

The %INCLUDE file EMPLOYEE.DAT contains the mapping instructions for the request EMPLOYEE_REQUEST.

```
RDU> CREATE REQUEST EMPLOYEE_REQUEST EMPLOYEE.DAT
```

EMPLOYEE.DAT contains a %INCLUDE instruction that points to a file containing the request text. For example:

```
$  TYPE EMPLOYEE.DAT
   FORM IS EMP_FORM;
   RECORD IS EMP_REC;
   %INCLUDE "DEPEND.DAT";
   END DEFINITION;
```

RDU checks the text in the %INCLUDE file for correct syntax and mapping instructions. When the EMPLOYEE_REQUEST is stored in the CDD, it contains the %INCLUDE instruction but not the text from the %INCLUDE file.

3.12 INPUT TO Instruction

Collects data from one or more form fields and returns it to one or more record fields.

Format

```
INPUT form-field TO   { record-field          }
                      { (record-field[,...]) }
        [,...];

INPUT %ALL;
```

Prompt

```
RDUDFN>
```

Instruction Parameters

form-field

The name of a field on the active form.

record-field

The name of a record field to which TDMS returns the form field data. You must specify preceding group field names only if they are necessary to make the reference unique. If you specify a list of record fields, you must enclose them in matching parentheses and separate the field names 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.

%ALL

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

Notes

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

If the operator does not enter data in a field mapped for input, TDMS returns one of the following to the receiving record field:

- Data mapped for output during the current request call
- Data in the form field from the immediately previous request call (if no data is output to the field during the current request call)
- Data associated with the form field by a form definition default (if no other data is in the field)

TDMS performs input mappings after it executes all output mappings.

When you use explicit syntax, RDU checks the form and record fields you specify in the INPUT TO instruction to see that:

- The fields exist in the records or forms used by a request
- The fields do not exist more than once in the records or form used by a request
- The mappings defined are valid (in the default Validate mode); that is, that the data types, structures, lengths, sign conditions, scale factors, and so on, of the fields 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 INPUT %ALL instruction if you want to collect data from all the fields on an active form and return the data to identically named record fields.

In a %ALL mapping, RDU does not create the *individual mappings* 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 mapping is not valid; that is, the data types, structures, lengths, scale factors, sign conditions, and so on of the fields are not compatible

RDU does, however, create (or replace, modify, or validate) 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 also appear in the log file if both the SET LOG command and the /LOG qualifier are specified.

Examples

```
RDUDFN> INPUT NAME TO BUFFER1.EMPLOYEE_NAME;
```

Allows the operator to enter data into the form field NAME and to return the data to the record field EMPLOYEE_NAME within the record BUFFER1.

```
RDUDFN> RECORDS ARE RECORD_1, RECORD_2;
RDUDFN>    INPUT NAME TO RECORD_2.EMPLOYEE_NAME,
RDUDFN>       BADGE   TO EMPLOYEE_BADGE,
RDUDFN>       SEX     TO EMPLOYEE_SEX;
```

Collects data from form fields NAME, BADGE, and SEX and returns it to record fields EMPLOYEE_NAME, EMPLOYEE_BADGE, and EMPLOYEE_SEX. The record field EMPLOYEE_NAME is in the record RECORD_2. The other two record fields may be in RECORD_2 or RECORD_1; RDU searches both records. The elements in the list of INPUT instructions are separated by commas. The list is terminated by a semicolon.

```
RDUDFN> FORM IS STATUS_FORM;
RDUDFN>     DISPLAY FORM STATUS_FORM;
RDUDFN>     INPUT %ALL;
```

Collects data from all the fields (that have identically named record fields) on the form STATUS_FORM and returns it to record fields of the same name.

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 PROGRAM KEY IS instruction.

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. Data entered on a keypad when the keypad is in Numeric mode cannot be received as a PRK. 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 in program request keys.

Keys you can define as program request keys in Application mode include:

- Digits 0-9, comma (,), period (.), hyphen (-)
- The combination of keystrokes: PF1 and any key on the main keyboard that represents a printable character (except the TAB key)

See the PROGRAM KEY IS instruction.

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.

If you define a keypad key as a PRK when TDMS is in Numeric mode, TDMS does not issue a warning message when it treats the data returned by a program request key as numeric data.

In a PRK, a key name phrase that begins with KEYPAD can be enabled only if you place the terminal keypad in Application mode with a KEYPAD IS APPLICATION instruction.

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.

3.14 [NO] LIGHT LIST Instruction

Turns keyboard lights off or on.

Format

```
[NO] LIGHT LIST number-list;
```

Prompt

```
RDUDFN>
```

Instruction Parameter

number-list

A list of numbers between 1 and 4, inclusive, separated by commas. Each number corresponds to one of four lights on the keyboard.

Notes

Specifies that one light, all lights, or a particular set of lights be turned off or on until you specify another LIGHT LIST or NO LIGHT LIST instruction.

You must specify the number of each light explicitly. To turn off all the lights, for instance, you must specify the numbers for all of the lights.

If a request turns a light on and does not turn it off again, it stays on even after the request completes.

Example

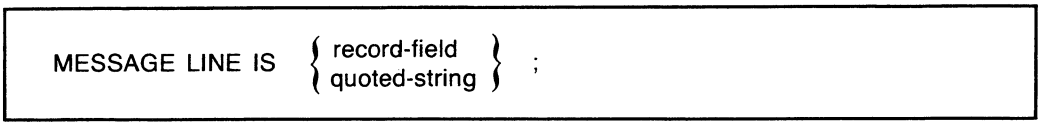
```
RDUDFN> LIGHT LIST 1, 2, 3;
RDUDFN> NO LIGHT LIST 4;
```

Lights keyboard lights 1,2,3 and then turns off light number 4. The operator sees lights 1, 2, and 3.

3.15 MESSAGE LINE IS Instruction

Writes the specified data to line 24 of the terminal.

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. 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 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

```
OUTPUT { record-field  } TO { form-field       }
       { quoted-string }    { (form-field[,...]) }
            [,...]    [WITH video-attribute[,...]];

OUTPUT %ALL;
```

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.

%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.

You can use the OUTPUT %ALL instruction if you want to display data in all the form fields on an active form that have 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 matching record field exists more than once in the records used by a request
- The mapping is not valid (field data types, structures, lengths, sign conditions, scale factors, and so on are not compatible)

RDU does, however, create (replace, modify, or validate) the request, unless *all* the mappings implied by the %ALL are incorrect.

If the /LOG qualifier is used, %ALL mappings will appear:

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

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

See the PROGRAM KEY IS instruction for how to use the OUTPUT TO instruction within a PRK.

Examples

```
RDUDFN>  OUTPUT EMPLOYEE_NAME TO NAME,
RDUDFN>             EMP_BADGE TO (NUMBER, BADGE);
```

Maps data from the EMPLOYEE_NAME record field to the form field NAME and data from the EMP_BADGE record field to the form fields NUMBER and BADGE.

```
RDUDFN>  OUTPUT EMP_NAME TO NAME,
RDUDFN>         EMP_BADGE TO BADGE,
RDUDFN>         EMP_SEX TO SEX;
```

Displays record fields EMP_NAME, EMP_BADGE, EMP_SEX in their respective form fields, NAME, BADGE, and SEX. Note that the list of record to form fields is separated by commas and terminated with a semicolon.

```
RDUDFN>  OUTPUT "BADGE_NO INCORRECT" TO (FIELD1, FIELD2, FIELD3);
```

Displays the text string in the three form fields named. Note that the receiving list of form fields is enclosed in parentheses.

```
RDUDFN>  OUTPUT  EMP_REC1.NAME_FIELD TO NAME,
RDUDFN>     BUFFER.ACCOUNTS_FIELD[3 TO 5] TO ACCOUNT[1 TO 3];
```

Outputs the record subfield NAME_FIELD within the record (or group field) EMP_REC1 to the form field NAME. Outputs the contents of entries 3 to 5 in the record array ACCOUNTS_FIELD to the first three fields in the form array or indexed form field ACCOUNT.

```
RDUDFN>  OUTPUT HIST_CHANGES.HIST_DEPT[1] TO DEPT[1],
RDUDFN>         HIST_CHANGES.HIST_DEPT[5] TO DEPT[5];

RDUDFN>  INPUT DEPT[1] TO HIST_DEPT[1],
RDUDFN>        DEPT[5] TO HIST_DEPT[5];
```

TDMS executes the OUTPUT mappings first. Note that you can refer to the field HIST_DEPT two ways, either by using the preceding group field name or by just specifying the field name.

```
RDUDFN>  OUTPUT CALENDAR_DAYS[5,4] TO CALENDAR[5,4];
```

TDMS displays a single element in the two-dimensional simple array CALENDAR_DAYS within the horizontally-indexed scrolled form field CALENDAR.

```
RDUDFN>  DISPLAY FORM PROJECT_FORM;
RDUDFN>     OUTPUT %ALL;
RDUDFN>     DESCRIPTION /*displays data in all fields
RDUDFN>                    on the form PROJECT_FORM
RDUDFN>                   (PROJECT_NO, PROJECT_LEADER,
RDUDFN>                   PROJECT_STATUS)*/;
```

Maps data from those record fields in the records used by a request that have identically named form fields on the active form.

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

```
PROGRAM KEY IS { GOLD   } "prk-name"  [[NO] CHECK;]
               { KEYPAD }

{ { OUTPUT quoted-string TO form-field [WITH video-attribute,...]; } }
{ { MESSAGE LINE IS quoted-string;                                 } }
{   RETURN quoted-string TO record-field;                          }

END PROGRAM KEY;
```

Prompt

```
RDUDFN>
```

Instruction Parameters

prk-name

The name of the program request key that you specify. You can choose one of the following:

- The word KEYPAD followed by any of these keypad keys:

  Digits 0-9
  Comma
  Period
  Hyphen

  The key must be enclosed in quotation marks.

- The word GOLD followed by any of the printable keyboard keys except the TAB key, including:

  A-Z
  a-z
  0-9
  The space bar

  Any of the set:

Ampersand (&)	Angle brackets (< >)	Asterisk (*)
At sign (@)	Caret (^)	Colon (:)
Comma (,)	Exclamation mark (!)	Dollar sign ($)
Equals sign (=)	Hyphen (-)	Parentheses (())
Percent sign (%)	Period (.)	Plus sign (+)
Pound sign (#)	Question mark (?)	Quotation marks (",')
Semicolon (;)	Slash (/)	Tilde (~)
Underscore (_)		

  The key must be enclosed in quotation marks. Note that uppercase and lowercase letters are received as the same key.

OUTPUT quoted-string TO form-field

Specifies a text string to be written to a form field when the operator presses the PRK named. You cannot specify both an OUTPUT text string and a MESSAGE LINE IS text string.

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 types the PRK named. 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 form field to which it is output. 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 the form field in which the text string is displayed when the operator presses the PRK.

record-field

The name of one field in a program record. Only one record field 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 also 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 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 in 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 INPUT 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 must specify Application mode for TDMS to execute a program request key using a keypad key. (See the KEYPAD MODE IS instruction.)

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

Examples

```
RDUDFN>  PROGRAM KEY IS GOLD "C"
RDUDFN>                 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 terminal 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
RDUDFN>                                    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 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.

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.

Notes

The records named must exist in the CDD if you are running RDU in Validate mode (the default mode). If RDU is not in Validate mode, it does not check for the existence of the specified record definition.

The RECORD IS instruction, if used, must appear in the header portion of a request.

You cannot define mappings to a record unless you identify that record in a RECORD IS instruction.

No two records in the RECORD IS instruction can have the same given name. If you want to use two record definitions in the CDD that have identical given names, you must specify a unique name using the WITH NAME phrase.

Note that the order of the record definition path names specified in the RECORD IS instruction *must* match the order of the parameters specified in the TSS$REQUEST call in the application program.

Examples

```
RDUDFN> RECORD IS EMPLOYEE_RECORD;
```

Specifies a record, EMPLOYEE_RECORD, using the given name.

```
RDUDFN> RECORD IS CDD$TOP.ACCOUNT.PAYROLL.EMPLOYEE_RECORD;
```

Specifies a record, CDD$TOP.ACCOUNT.PAYROLL.EMPLOYEE_RECORD, using the full path name.

```
RDUDFN> RECORD IS CDD$TOP.ACCOUNT.EMPLOYEE WITH NAME ACC_RECORD;
RDUDFN> RECORD IS CDD$TOP.FINANCE.EMPLOYEE WITH NAME FIN_RECORD;
```

Specifies two records, CDD$TOP.ACCOUNT.EMPLOYEE and CDD$TOP.FINANCE.EMPLOYEE, using the full path names and the WITH NAME modifier. In subsequent references to these records within this request, use the unique names ACC_RECORD and FIN_RECORD.

```
RDUDFN> RECORDS ARE EMPLOYEE_REC, ACCOUNTING_REC, PERSONNEL_REC;
```

Specifies several records using the given names.

3.19 REQUEST IS Instruction

Identifies the request or requests RDU will place in a request library file during the build operation.

Format

```
{ REQUEST IS  }
{ REQUESTS ARE } request-path-name [WITH NAME unique-request-name],...;
```

Prompt

```
RDUDFN>
```

Instruction Parameter

request-path-name

The CDD path name (given, relative, or full) of an existing request definition. Note that the name you use for 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 requests, or a request and a form, cannot use the same name within the body of a request library definition. If two request given names, or a request 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-request-name

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

The unique request name, if specified, is the one the TDMS application program must use in the TSS$REQUEST call. If the unique name is not specified, you use the given name.

Notes

The REQUEST IS instruction is one of two instructions that usually make up the text of a request library definition; the only required instruction is END DEFINITION. You can also include a FILE IS instruction and one or more FORM IS instructions in a request library definition.

The REQUEST IS instruction specifies the request that RDU extracts from the CDD and places in the request library file during the build operation.

If the specified request does not exist in the CDD and RDU is in Validate mode, RDU does not create a request library definition in the CDD. Instead, it issues an error message. You must create the request or set RDU to Novalidate mode.

Examples

```
RDUDFN>  REQUEST IS ACCOUNTING_REQ ;
RDUDFN>  REQUEST IS EMPLOYEE_REQ;
RDUDFN>  FILE IS "EMPINFO.RLB";
RDUDFN>  END DEFINITION;
```

Specifies two requests to be placed in the request library file EMPINFO.RLB.

```
RDUDFN>  REQUEST IS FINANCE.PAYROLL_EMPLOYEE WITH NAME FIN_REQUEST;
RDUDFN>  REQUEST IS ACCOUNTING.PAYROLL_EMPLOYEE WITH NAME ACCT_REQUEST;
RDUDFN>  END DEFINITION;
```

Specifies two requests to place in a request library file. Note that although these two requests have different CDD path names, the given names are the same. By using the WITH NAME modifier, you provide a way to refer to these requests uniquely within an application program. The TDMS application uses these unique names to refer to these requests.

3.20 [NO] RESET FIELD Instruction

Allows you to reset the video attributes of a field in an active form to the defaults in the form definition.

Format

```
[NO] RESET 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 series of form fields separated by commas.

%ALL

All the fields on the active form.

Notes

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

- Any video instruction executed in the base or outer conditional instruction.
- Any video instruction that is still active from a previous request call. A video instruction is still active when both of these conditions are present:
  - 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.

Only the video attributes are changed, not the text in the fields.

Example

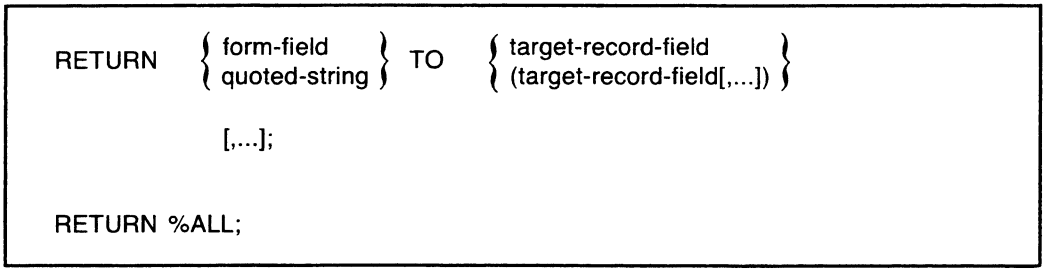
```
RDUDFN> RESET FIELD NAME, BADGE, SEX;
```

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.

%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.

Only the Reverse or No Reverse attribute of the form field is affected. All other video attributes of the field and the content of the field remain unchanged. For instance, if you specify a BLINK instruction in addition to a REVERSE instruction, the form field is both reversed and blinking.

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.

3.23 [NO] RING BELL Instruction

Signals the operator when TDMS begins executing a request.

Format

```
[NO] RING BELL [number-of-rings];
```

Prompt

```
RDUDFN>
```

Instruction Parameter

number-of-rings

The number of times you want TDMS to ring the terminal bell. The number must be a positive integer and cannot exceed 255.

Notes

If you do not specify a number following the RING BELL instruction, TDMS rings the bell once when it begins to execute a request.

If you specify a number following the NO RING BELL instruction, RDU will signal a syntax error.

Examples

```
RDUDFN> RING BELL 2;
```

Rings the terminal bell twice when TDMS begins executing the request.

```
RDUDFN> RING BELL;
```

Rings the terminal bell once when TDMS begins executing the request.

3.24 SIGNAL [MODE] IS Instruction

Specifies that the terminal use its bell or a reverse screen to signal the operator.

Format

```
SIGNAL [MODE] IS { BELL    } ;
                 { REVERSE }
```

Prompt

```
RDUDFN>
```

Instruction Parameters

BELL

The terminal bell.

REVERSE

The reverse video screen.

Notes

Use this instruction in conjunction with the SIGNAL OPERATOR instruction.

When you specify the instruction SIGNAL OPERATOR or when the operator makes an error, TDMS either:

- Reverses the video of the screen momentarily (if the Signal mode is set to REVERSE)
- Rings the terminal bell to signal the operator (by default or if the Signal mode is set to BELL)

Once the Signal mode is set to either BELL or REVERSE, the Signal mode remains the same until it is explicitly changed by another request.

Note that on VT52 terminals, all signals to the operator ring the terminal bell, regardless of how you set the Signal mode.

Examples

```
RDUDFN> SIGNAL MODE IS REVERSE;
```

Sets the Signal mode so that TDMS will reverse the screen video when the operator makes an error or when the operator is signaled by a SIGNAL OPERATOR instruction.

```
RDUDFN> SIGNAL IS BELL;
```

Sets the Signal mode so that TDMS will ring a bell when the operator makes an error or when the operator is signaled by a SIGNAL OPERATOR instruction.

3.25 [NO] SIGNAL OPERATOR Instruction

Signals the operator by ringing the terminal bell or reversing the screen video when TDMS begins executing a request.

Format

```
[NO] SIGNAL OPERATOR;
```

Prompt

```
RDUDFN>
```

Notes

Use the SIGNAL MODE IS instruction to determine whether the reversed screen or the terminal bell is used to signal the operator.

Note that on VT52 terminals, all signals to the terminal operator ring the terminal bell, regardless of how you set Signal mode in a request.

Examples

```
RDUDFN> SIGNAL MODE IS REVERSE;
RDUDFN> SIGNAL OPERATOR;
```

Signals the beginning of request execution by reversing the screen momentarily.

```
RDUDFN> NO SIGNAL OPERATOR;
```

Stops TDMS from ringing the terminal bell that signals the beginning of request execution.

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

Only the Underline or No Underline attribute of the form field is affected. All other video attributes of the field and the content of the field remain unchanged. For instance, if you specify an UNDERLINE instruction in addition to a BLINK instruction, the field is both underlined and blinking.

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.

3.27 USE FORM Instruction

Displays a form using the form background and field contents from the immediately previous request call. Sets the active form for the request.

Format

```
USE FORM form-name [WITH OFFSET offset-value];
```

Prompt

```
RDUDFN>
```

Instruction Parameters

form-name

The form name can be either:

- The given name of an existing form definition.
- The unique form name specified in the WITH NAME clause of the FORM IS instruction. If the WITH NAME clause is specified in the FORM IS instruction, you *must* use it in the USE FORM instruction.

WITH OFFSET offset-value

The keywords WITH OFFSET and a number that specifies a beginning line at which the form is displayed on the screen. The offset value is added to the number of the line where the form would be displayed without the WITH OFFSET modifier. The offset value is a value between 0 and +22. The plus sign preceding the number is optional.

When you validate the request or build the request library file, if the offset value you specify would cause any portion of the form to be displayed past the twenty-third line of the screen, RDU issues an error level message and does not build the request library file.

Notes

The form you specify in the USE FORM instruction is the active form; that is, it is the only form to and from which data can be mapped during that call to the request.

TDMS displays the form context (background and field contents) from the previous request call only if the form was used in that request call in a DISPLAY FORM or USE FORM instruction.

If you did not use that form, TDMS has no form context and executes the USE FORM instruction as though it were a DISPLAY FORM instruction.

If you specify output mappings or video changes with a USE FORM instruction, the output mappings override the form context from the previous request call.

Examples

```
RDUDFN>  USE FORM EMPFORM WITH OFFSET 2;
```

TDMS displays the form EMPFORM, beginning at line 3, with the contents and field attributes from the immediately previous request call. If EMPFORM was not used in the immediately previous request call, TDMS displays the form with its form definition defaults.

```
RDUDFN>  USE FORM EMPLOYEE_FORM;
RDUDFN>  OUTPUT NAME TO NAME;
```

TDMS displays the form EMPLOYEE_FORM with the contents and field attributes from the immediately previous request call. The data output to the form field NAME, however, overrides the contents saved from the previous call to the request.

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:

- The ENTER or RETURN key
- A PRK

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.

TDMS Synchronous Programming Calls 4

This chapter provides a complete description of all the TDMS synchronous programming calls. Calls are listed in alphabetical order.

The discussion of each call includes:

Format	The generic description of the call. See Table 4-1 for an explanation of the notation.
Call Parameters	The description of the arguments passed.
Return Status	A general explanation of return status and a list of the codes that can be returned on that call. In addition, you see the error severity code associated with the return status. See Table 4-2 for the error severity codes and their explanations.
Note	Additional information on using the call.
Examples	In VAX BASIC, VAX COBOL, and VAX FORTRAN.

At the end of this chapter, there are table summaries of all TDMS calls by language:

- Table 4-4 in VAX BASIC
- Table 4-5 in VAX COBOL
- Table 4-6 in VAX FORTRAN

4.1 Notation Used in This Chapter

Table 4-1 gives a complete list of the data types and passing mechanisms used for TDMS programming calls.

Table 4-1: Parameter Passing Notation

Notation	Explanation
mz.r	Modifiable unspecified type, by reference
rlu.r	Read-only unsigned longword logical type, by reference
rlu.v	Read-only unsigned longword logical type, by value
rt.dx	Read-only character string, class type in descriptor
szem.r	Call without stack unwinding, procedure entry mask, by reference
wlc.v	Write-only longword return status, by value
wlu.r	Write-only unsigned longword logical type, by reference
wt.dx	Write-only character string, class type in descriptor
wwu.r	Write-only unsigned word logical type, by reference

All TDMS programming calls return a standard VAX/VMS status code. Each status code has an associated severity level. Table 4-2 explains the possible severity levels of the status codes that TDMS programming calls return.

Table 4-2: Error Severity Codes for Return Status

Error Severity	Explanation
S	Successful completion
I	Information only
F	Indicates a Severe error; execution of the program does not continue, and the procedure does not produce any output

4.2 TSS$CANCEL Call

Cancels all input/output operations in progress on a channel.

Format

```
ret-status.wlc.v = TSS$CANCEL(channel.rlu.r)
```

Call Parameter

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN call; you pass this parameter by reference.

Return Status

Ret-status is the standard VAX/VMS return status indicating the success or failure of the call. The codes the TSS$CANCEL call can return are:

TSS$_BUGCHECK

Fatal internal software error (F)

TSS$_CANINPROG

Cancel in progress on channel (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_NOIO

No I/O in progress (I)

TSS$_NORMAL

Normal successful completion (S)

Notes

The TSS$CANCEL call is the only way in a TDMS application to cancel a TDMS input/output call in progress. For example, if you have an application control mailbox that receives messages from an asynchronous $QIO system service call, the AST routine could then cancel a TDMS call that is in progress.

If you cancel a TDMS input/output call, no further characters typed on the terminal are processed until you issue another call to TSS$REQUEST or TSS$READ_MSG_LINE.

The cancel is complete when the canceled call returns to the application program, not when TSS$CANCEL returns to the application program.

Calls that can be canceled are:

- TSS$COPY_SCREEN and TSS$COPY_SCREEN_A
- TSS$READ_MSG_LINE and TSS$READ_MSG_LINE_A
- TSS$REQUEST and TSS$REQUEST_A
- TSS$WRITE_MSG_LINE and TSS$WRITE_MSG_LINE_A

Note that TSS$CANCEL is the only synchronous TDMS call that can be used at the asynchronous system trap (AST) level without causing an error. Note also that you can mix synchronous and asynchronous calls. For example, you can use TSS$OPEN_A with TSS$CLOSE, or TSS$DECL_AFK_A with TSS$UNDECL_AFK.

Examples

BASIC

```
Return_status = TSS$CANCEL(Channel)
```

COBOL

```
CALL "TSS$CANCEL"
     USING BY REFERENCE Channel,
     GIVING Return-status.
```

FORTRAN

```
Return_status = TSS$CANCEL(%REF(Channel))
```

4.3 TSS$CLOSE Call

Closes the channel to the terminal and optionally clears the screen.

Format

```
ret-status.wlc.v = TSS$CLOSE(channel.rlu.r
                             [,clear-screen.rlu.r])
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN call; you pass this parameter by reference.

clear-screen

The address of the longword that specifies whether or not the screen is to be cleared; you pass this parameter by reference. If the value of this parameter is 1, TDMS clears the screen. The clear-screen parameter is optional and, if it is omitted or if the parameter is 0, the screen is not cleared.

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$_BADFLAGS

Flag parameter has invalid value (F)

TSS$_BUGCHECK

Fatal internal software error (F)

TSS$_CANINPROG

Cancel in progress on channel (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_IOINPROG

I/O in progress on channel (F)

TSS$_NORMAL

Normal successful completion (S)

TSS$_SYNASTLVL

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

Notes

You should close the TDMS channel when you are finished using TDMS for input/output on that terminal. As a result:

- TDMS resources associated with that terminal are released.
- Screen video characteristics are reset.
- The cursor is returned to the bottom left hand corner of the terminal screen.
- The keypad is reset to Numeric mode.

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$CLOSE(Channel, &
                          Clear_screen)
```

COBOL

```
CALL "TSS$CLOSE"
     USING BY REFERENCE Channel,
           BY REFERENCE Clear-screen,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$CLOSE(%REF(Channel),
1                          %REF(Clear_screen))
```

4.4 TSS$CLOSE_RLB Call

Closes a request library file that is currently open.

Format

```
ret-status.wlc.v = TSS$CLOSE_RLB(library-id.rlu.r)
```

Call Parameter

library-id

The address of the longword containing the unique number that identifies the library to close; you pass this parameter by reference. It must be the same number that was assigned by TSS$OPEN_RLB.

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$_INVARG

Invalid arguments (F)

TSS$_INVRLBID

Invalid request library ID (F)

TSS$_NORMAL

Normal successful completion (S)

TSS$_RLBINUSE

Request library in use (F)

TSS$_SYNASTLVL

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

Notes

You can close and reopen the same request library file in a program, as needed.

You can have multiple request library files open at the same time. Each request library file requires its own TSS$OPEN_RLB and TSS$CLOSE_RLB calls.

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$CLOSE_RLB(Library_id)
```

COBOL

```
CALL "TSS$CLOSE_RLB"
     USING BY REFERENCE Library-id,
     GIVING Return-status.
```

FORTRAN

```
Return_status = TSS$CLOSE_RLB(%REF(Library_id))
```

4.5 TSS$COPY_SCREEN Call

Copies the current contents of the screen to the file specified or the file defined by the logical TSS$HARDCOPY.

Format

```
ret-status.wlc.v = TSS$COPY_SCREEN(channel.rlu.r
                                   ,[file-spec.rt.dx]
                                   [,append-flag.rlu.r])
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN call; you pass this parameter by reference.

file-spec

The address of a string descriptor pointing to the VMS file specification to which the contents of the screen will be directed. You pass this optional parameter by descriptor.

If this parameter is not specified, the value of the logical TSS$HARDCOPY is used to determine where the contents of the screen are directed.

If this parameter is not specified and the logical TSS$HARDCOPY is not defined, this call does nothing. The return status is TSS$_NOOUTFILE, an informational status.

append-flag

The address of a longword that specifies whether to create a new version of the file or to append the copy of the screen contents to the latest version of the file. You pass this optional parameter by reference. If present, this parameter must have a value of either 0 or 1.

A new version of the output file will be created if:

- The parameter is not present
- The flag has a value of 0
- There is no existing version of the file

Otherwise, the current contents of the screen will be appended to the latest version of an existing VMS file.

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$_BADFLAGS

Flag parameter has invalid value (F)

TSS$_BUGCHECK

Fatal internal software error (F)

TSS$_CANCEL

Call canceled by TSS$CANCEL (F)

TSS$_CANINPROG

Cancel in progress on channel (F)

TSS$_COPYOUTERR

TSS$COPY_SCREEN output error (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_IOINPROG

I/O in progress on channel (F)

TSS$_NOOUTFILE

No output file definition found (I)

TSS$_NORMAL

Normal successful completion (S)

TSS$_SYNASTLVL

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

Notes

TSS$COPY_SCREEN supplies a callable PF4 (hardcopy) function to the program. You cannot issue a TSS$COPY_SCREEN call while input or output is active on the specified channel, such as with an outstanding TSS$REQUEST or TSS$WRITE_MSG_LINE call.

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$COPY_SCREEN(Channel, &
                                File_spec, &
                                Append_flag)
```

COBOL

```
CALL "TSS$COPY_SCREEN"
     USING BY REFERENCE Channel,
           BY DESCRIPTOR File-spec,
           BY REFERENCE Append-flag,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$COPY_SCREEN(%REF(Channel),
1                                %DESCR(File_spec),
2                                %REF(Append_flag))
```

4.6 TSS$DECL_AFK Call

Enables the operator's use of an application function key (AFK) by specifying the AFK and associating it with a service routine and/or event flag.

Format

```
ret-status.wlc.v = TSS$DECL_AFK(channel.rlu.r
                                ,key-id.rlu.r
                                {,key-efn.rlu.r                  }
                                {                                }
                                {,key-astadr.szem.r              }
                                { [,key-astprm.rlu.v]            }   )
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN call; you pass this parameter by reference.

key-id

The address of a longword that contains a code representing the AFK; you pass this parameter by reference. When the operator presses the key represented by the key-id parameter, the application program is notified.

Because the key-id parameter is a code, each call to TSS$DECL_AFK can activate only one key. Table 4-3 lists the TDMS application function keys.

Table 4-3: 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 set when the operator presses the AFK; 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 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 may use the AST service routine with or without an AST parameter. You may also use the AST service routine with an event flag.

key-astprm

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

TDMS treats this parameter as a value: you can pass any type of parameter you would like your AST routine to receive, including addresses (parameters by reference).

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

When an AFK is pressed, TDMS does one of the following:

- Sets the event flag specified for this AFK (if any)
- Calls the service routine (if any) at AST level and passes it:
  - The AST parameter
  - The TDMS channel on which it was pressed
  - The key-id parameter of the AFK that was pressed
- Both sets the event flag and calls the service routine

  In this case, TDMS sets the event flag before it calls the service routine.

TDMS calls the service routine whether or not there is an outstanding request, as soon as the VMS terminal driver processes the key stroke.

All the control keys except CTRL/Q and CTRL/S may be defined as AFKs.

Note that some control keys (for example, CTRL/H or BACK SPACE, CTRL/I or TAB, CTRL/J or LINE FEED, and CTRL/M or RETURN) may be defined as AFKs. You should be careful when designing functions for AFKs that already have a special meaning for TDMS; otherwise, unexpected results might be generated. For example, if you define CRTL/M (RETURN) as an AFK, then you must provide another way of terminating the request.

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 Key_ast_routine
              .
              .
              .
Return_status = TSS$DECL_AFK(Channel, &
                             Key_id, &
                             Key_event_flag_number, &
                             Key_ast_routine,       &
                             Key_ast_parameter BY VALUE)
```

TSS$DECL_AFK

COBOL

```
CALL "TSS$DECL_AFK"
     USING BY REFERENCE Channel,
           BY REFERENCE Key-id,
           BY REFERENCE Key-event-flag-number,
           BY REFERENCE Key-ast-routine,
           BY VALUE Key-ast-parameter,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$DECL_AFK(%REF(Channel),
1                             %REF(Key_id),
2                             %REF(Key_event_flag_number),
3                             %REF(Key_ast_routine),
4                             Key_ast_parameter)
```

4.7 TSS$OPEN Call

Opens a channel to a terminal for input and output.

Format

```
ret-status.wlc.v = TSS$OPEN(channel.wlu.r
                            [,device.rt.dx])
```

Call Parameters

channel

The address of a longword to receive the TDMS channel number; you pass this parameter by reference.

device

The address of a string descriptor pointing to the device name of the terminal; you pass this optional parameter by descriptor. This can be a logical name or a physical device specification. This parameter is optional; the default is SYS$INPUT.

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$_ERROPNDEV

Error opening device (F)

TSS$_ILLDEVCHAR

Illegal device characteristics (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_NORMAL

Normal successful completion (S)

TSS$_SYNASTLVL

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

Notes

TDMS assigns a unique channel number to identify the terminal. You use this channel number as a parameter in other TDMS calls to identify what terminal to use for the operation.

Input and output must be performed on the same terminal; therefore, the input/output streams cannot be split between two terminals. For example, if you open a channel to SYS$INPUT, and you redefine SYS$OUTPUT as some other device, both input and output still occur on SYS$INPUT.

The channel number returned is not the VAX/VMS channel number returned by the SYS$ASSIGN service. Input/output calls to other systems (such as VAX RMS or $QIO) *should not* be issued to the terminal. If you try to issue calls to other systems, you might generate unexpected results.

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$OPEN(Channel,     &
                         Device_name)
```

COBOL

```
CALL "TSS$OPEN"
     USING BY REFERENCE Channel,
           BY DESCRIPTOR Device-name,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$OPEN(%REF(Channel),
1                         %DESCR(Device_name))
```

4.8 TSS$OPEN_RLB Call

Opens a request library file containing the requests to be used in the application program.

Format

```
ret-status.wlc.v = TSS$OPEN_RLB(rlb-file-spec.rt.dx
                                ,library-id.wlu.r)
```

Call Parameters

rlb-file-spec

The address of a string descriptor pointing to the VMS file specification of the request library file; you pass this parameter by descriptor. The file specification may include logical names. The default file type is .RLB.

library-id

The address of the longword to receive the unique number identifying the request library file; you pass this parameter by reference.

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$_ERROPNRLB

Error opening request library (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_NORMAL

Normal successful completion (S)

TSS$_SYNASTLVL

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

Notes

TDMS assigns a unique library ID to identify the request library file. You use this library ID as a parameter in TDMS calls that require a library ID.

Request library files, built with RDU, contain a group of requests to use for a particular application.

A request library file must be open before you can issue a successful TSS$REQUEST call. You can have multiple request library files open at the same time. Each request library file requires its own TSS$OPEN_RLB and TSS$CLOSE_RLB calls.

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$OPEN_RLB(Request_library_file, &
                             Library_id)
```

COBOL

```
CALL "TSS$OPEN_RLB"
     USING BY DESCRIPTOR Request-library-file,
           BY REFERENCE Library-id
     GIVING Return-status.
```

TSS$OPEN_RLB

FORTRAN

```
 Return_status = TSS$OPEN_RLB(%DESCR(Request_library_file),
1                             %REF(Library_id))
```

4.9 TSS$READ_MSG_LINE Call

Writes a prompt on the terminal on the reserved message line (optionally), and reads a line of input from the operator.

Format

```
ret-status.wlc.v = TSS$READ_MSG_LINE(channel.rlu.r
                                     ,response-text.wt.dx
                                     ,[message-prompt.rt.dx]
                                     [,response-length.wwu.r])
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN call; you pass this parameter by reference.

response-text

The address of a static or dynamic string descriptor pointing to the buffer that receives the text entered by the operator; you pass this parameter by descriptor.

message-prompt

The address of a string descriptor pointing to the prompt you want to display on the message line; you pass this optional parameter by descriptor.

response-length

The address of a word to receive the length of the message line read into the buffer; you pass this optional parameter by reference.

This parameter is useful for setting the length of the response text so that it can be compared with other strings within the application program. If you use a static string descriptor for the response-text parameter and omit the response-length parameter, you cannot tell how long the response text is. A comparison might fail because the two strings are unequal in length.

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$_CANCEL

Call canceled by TSS$CANCEL (F)

TSS$_CANINPROG

Cancel in progress on channel (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_INVTXTLEN

Invalid text length (F)

TSS$_IOINPROG

I/O in progress on channel (F)

TSS$_NONPRICHA

Field contains non-printable characters (F)

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 be up to 132 characters. When the operator presses the RETURN key or any other request processing key, the message line is cleared.

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))
```

4.10 TSS$REQUEST Call

Executes the request named in the call.

Format

```
ret-status.wlc.v = TSS$REQUEST(channel.rlu.r
                               ,library-id.rlu.r
                               ,request-name.rt.dx
                               [,record1.mz.r
                               [,record2.mz.r
                                  .
                                  .
                                  .
                               [,recordn.mz.r]]])
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned by the TSS$OPEN call; you pass this parameter by reference.

library-id

The address of the longword containing the unique number that identifies the library containing the desired request; you pass this parameter by reference. It must be the same number that was assigned by TSS$OPEN_RLB.

request-name

The address of a string descriptor pointing to the name of the request to use; you pass this parameter by descriptor.

record(s)

The address of the record(s) that the request uses for mapping data between the form and the program record; you pass this optional parameter by reference. If you specify more than one record, the order of the records must be the same as in the RECORD IS instruction(s) in the request.

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$_CANCEL

Call canceled by TSS$CANCEL (F)

TSS$_CANINPROG

Cancel in progress on channel (F)

TSS$_CONVERR

Data type conversion error (F)

TSS$_DISFORERR

Form display failed (F)

TSS$_ILLDEVCHAR

Illegal device characteristics (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_MULTFORM

Multiple active forms are illegal (F)

TSS$_NORMAL

Normal successful completion (S)

TSS$_PRKCHECK

Request was terminated by a check PRK (I)

TSS$_PRKNOCHECK

Request was terminated by a nocheck PRK (I)

TSS$_REQMAJVERMIS

Request binary version mismatch on major id (F)

TSS$_REQMINVERMIS

Request binary version mismatch on minor id (F)

TSS$_REQNOTFOU

Request definition not found (F)

TSS$_SYNASTLVL

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

Notes

The request must be in a currently open request library file.

If your programming language supports the extraction of records from the CDD, record structure should not be a problem because you can use the same record definition in your program that was used when the request library file was built. Note that if you use CDDL or DATATRIEVE to define your records, you must pass a variable name to TSS$REQUEST that corresponds to the top level structure name in the record definition. With languages that do not support the CDD, you must be careful to define your records so that they are compatible with your programming language. Otherwise you will generate unexpected results.

The order of the records passed to TDMS must match the order of the records in the RECORD IS instruction in the request.

Because you pass records by reference, TDMS has no way of validating that you are passing the correct records.

The order in which instructions are specified in the request has no effect on the order in which TDMS executes the instructions. TDMS executes request instructions in the following order (if they are present in the request):

1. Control field evaluation.
2. Output mappings from the program record to the form, including data type conversion to text format for display, default field instructions, and video instructions.
3. Display of the form.
4. Input mappings from the terminal to the program, including data type conversion from text to the data type of the record fields.
5. Return operations.

The following form definition features cannot be used on a VT52 terminal:

- 132-column screen
- Scrolled region

You will generate a run-time error *only when* you try to display a form definition containing one of those features. For example, if you run an application on a VT52 and use form definitions without those features for the first five request calls, all of those calls will be successful. But if the sixth request call contains a form definition with one of those features, it will fail with the message:

%TSS-F-ILLDEVCHAR, illegal device characteristics

You should either change the form definition or run the application on a VT100-compatible terminal.

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$REQUEST(Channel, &
                            Library_id, &
                            Request_name, &
                            Record_1, &
                            Record_2, &
                            Record_n)
```

COBOL

```
CALL "TSS$REQUEST"
     USING BY REFERENCE Channel,
           BY REFERENCE Library-id,
           BY DESCRIPTOR Request-name,
           BY REFERENCE Record-1,
           BY REFERENCE Record-2,
           BY REFERENCE Record-n,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$REQUEST(%REF(Channel),
1                            %REF(Library_id),
2                            %DESCR(Request_name),
3                            %REF(Record_1),
4                            %REF(Record_2),
5                            %REF(Record_n))
```

4.11 TSS$SIGNAL Call

Signals the return status and any extended status from the immediately preceding TDMS call.

Format

```
ret-status.wlc.v = TSS$SIGNAL
```

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$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F) - (You must pass a null parameter list)

TSS$_NORMAL

Normal successful completion (S)

TSS$_SYNASTLVL

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

Notes

This call refers to the immediately preceding TDMS call in an application program. If you issue TSS$SIGNAL as the first call in your program, it will signal TSS$_NORMAL.

Note

If you use a programming language that does not let you pass 0 parameters to a procedure when you use it as a function call, you must include the null parameter list at the end of the call. See the FORTRAN example in this section.

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$SIGNAL
```

COBOL

```
CALL "TSS$SIGNAL" GIVING Return-status.
```

FORTRAN

```
Return_status = TSS$SIGNAL()
```

4.12 TSS$TRACE_OFF Call

Turns off the TDMS Trace facility.

Format

```
ret-status.wlc.v = TSS$TRACE_OFF
```

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$_INVARG

Invalid arguments (F)

TSS$_NORMAL

Normal successful completion (S)

TSS$_SYNASTLVL

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

TSS$_TRAOFF

Trace already off (I)

Notes

A message is written to the trace output file, indicating that the Trace facility is turned off.

Note

If you use a programming language that does not let you pass 0 parameters to a procedure when you use it as a function call, you must include the null parameter list at the end of the call. See the FORTRAN example in this section.

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$TRACE_OFF
```

COBOL

```
CALL "TSS$TRACE_OFF" GIVING Return-status.
```

FORTRAN

```
Return_status = TSS$TRACE_OFF()
```

4.13 TSS$TRACE_ON Call

Turns on the TDMS Trace facility, which traces the execution of a request at run time and traces TDMS calls.

Format

```
ret-status.wlc.v = TSS$TRACE_ON
```

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$_ERROPNTRA

Error opening trace log file (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_NORMAL

Normal successful completion (S)

TSS$_SYNASTLVL

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

TSS$_TRAON

Trace already on (I)

Notes

If the logical name TSS$TRACE_OUTPUT is defined, messages are written to a trace output file, which can be any file specification. The default file type is .LOG.

Otherwise, the trace output defaults to DBG$OUTPUT, which in turn defaults to SYS$OUTPUT.

You can define the logical name, TSS$TRACE_OUTPUT, before you run an application program. If you do, the Trace facility is automatically turned on when the first TDMS call is issued in the application program.

Note

If you use a programming language that does not let you pass 0 parameters to a procedure when you use it as a function call, you must include the null parameter list at the end of the call. See the FORTRAN example at the end of this section.

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$TRACE_ON
```

COBOL

```
CALL "TSS$TRACE_ON" GIVING Return-status.
```

FORTRAN

```
Return_status = TSS$TRACE_ON()
```

4.14 TSS$UNDECL_AFK Call

Disables an application function key (AFK) and its associated service routine and/or event flag.

Format

```
ret-status.wlc.v = TSS$UNDECL_AFK(channel.rlu.r
                                  ,key-id.rlu.r)
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN call; you pass this parameter by reference.

key-id

The address of a longword containing the code that represents the AFK that is no longer needed by the application program. You pass this parameter by reference.

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

You use this call to deactivate asynchronous notification of an AFK. After the TSS$UNDECL_AFK call is issued, TDMS no longer calls the service routine specified in the matching TSS$DECL_AFK call. That is, after the TSS$UNDECL_AFK call, the program is no longer notified when the operator presses the key.

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$UNDECL_AFK(Channel, &
                               Key_id)
```

COBOL

```
CALL "TSS$UNDECL_AFK"
     USING BY REFERENCE Channel,
           BY REFERENCE Key-id,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$UNDECL_AFK(%REF(Channel),
1                               %REF(Key_id))
```

4.15 TSS$WRITE_BRKTHRU Call

Writes a message to the reserved message line on the screen, interrupting the current request or message line operation in order to do so.

Format

```
ret-status.wlc.v = TSS$WRITE_BRKTHRU(channel.rlu.r
                                     ,message-text.rt.dx
                                     [,bell-flag.rlu.r])
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN call; you pass this parameter by reference.

message-text

The address of a string descriptor pointing to the text to be displayed on the message line; you pass this parameter by descriptor.

bell-flag

The address of a longword containing a flag for the terminal bell; you pass this optional parameter by reference. If the parameter is set to 1, the flag causes the terminal bell to ring when the message text is displayed. If you do not pass this parameter or if this parameter has a value of 0, TDMS does not ring the bell.

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$_BADFLAGS

Flag parameter has invalid value (F)

TSS$_BUGCHECK

Fatal internal software error (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_NONPRICHA

Field contains non-printable characters (F)

TSS$_NORMAL

Normal successful completion (S)

TSS$_SYNASTLVL

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

Notes

The message text for this call is truncated if its length is greater than the current terminal line size.

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$WRITE_BRKTHRU(Channel, &
                                  Message_text, &
                                  Bell_flag)
```

TSS$WRITE_BRKTHRU

COBOL

```
CALL "TSS$WRITE_BRKTHRU"
     USING BY REFERENCE Channel,
           BY DESCRIPTOR Message-text,
           BY REFERENCE Bell-flag,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$WRITE_BRKTHRU(%REF(Channel),
1                                  %DESCR(Message_text),
2                                  %REF(Bell_flag))
```

4.16 TSS$WRITE_MSG_LINE Call

Writes a message to the reserved message line on the terminal.

Format

```
ret-status.wlc.v = TSS$WRITE_MSG_LINE(channel.rlu.r
                                      ,message-text.rt.dx)
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN call; you pass this parameter by reference.

message-text

The address of a string descriptor pointing to the text to be displayed on the message line; you pass this parameter by descriptor.

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$_CANCEL

Call canceled by TSS$CANCEL (F)

TSS$_CANINPROG

Cancel in progress on channel (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_INVTXTLEN

Invalid text length (F)

TSS$_NONPRICHA

Field contains non-printable characters (F)

TSS$_NORMAL

Normal successful completion (S)

TSS$_SYNASTLVL

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

Notes

TDMS clears the reserved message line before it displays the message on the terminal. You can use this line to let the operator know about the status of AFKs.

TDMS uses the message line to display error messages; therefore, it is uncertain how long the line will be displayed. The cases when the message line is cleared are:

- If the operator makes an error on input
- If the operator exits the current input field
- When the screen is cleared

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$WRITE_MSG_LINE(Channel, &
                                   Message_text)
```

COBOL

```
CALL "TSS$WRITE_MSG_LINE"
     USING BY REFERENCE Channel,
           BY DESCRIPTOR Message-text,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$WRITE_MSG_LINE(%REF(Channel),
1                                   %DESCR(Message_text))
```

Table 4-4 lists the VAX TDMS synchronous calls in VAX BASIC. The calls are in alphabetical order.

Table 4-4: TDMS Synchronous Programming Calls in VAX BASIC

Call	Example
TSS$CANCEL	RET_STATUS = TSS$CANCEL(CHANNEL)
TSS$CLOSE	RET_STATUS = TSS$CLOSE & (CHANNEL, & CLEAR_SCREEN)
TSS$CLOSE_RLB	RET_STATUS = TSS$CLOSE_RLB & (LIBRARY_ID)
TSS$COPY_SCREEN	RET_STATUS = TSS$COPY_SCREEN & (CHANNEL, & FILE_SPEC, & APPEND_FLAG)
TSS$DECL_AFK *	RET_STATUS = TSS$DECL_AFK & (CHANNEL, & KEY_ID, & KEY_EVENT_FLAG_NUMBER, & KEY_AST_ROUTINE, & KEY_AST_PARAMETER BY VALUE)
TSS$OPEN	RET_STATUS = TSS$OPEN(CHANNEL, & DEVICE)
TSS$OPEN_RLB	RET_STATUS = TSS$OPEN_RLB & (REQUEST_LIBRARY_FILE, & LIBRARY_ID)
TSS$READ_MSG_LINE	RET_STATUS = TSS$READ_MSG_LINE & (CHANNEL, & RESPONSE_TEXT, & MESSAGE_PROMPT, & RESPONSE_LENGTH)
TSS$REQUEST	RET_STATUS = TSS$REQUEST & (CHANNEL, & LIBRARY_ID, & REQUEST_NAME, & RECORD_1, & RECORD_2, & RECORD_n)

(continued on next page)

Table 4-4: TDMS Synchronous Programming Calls in VAX BASIC (Cont.)

Call	Example
TSS$SIGNAL	RET_STATUS = TSS$SIGNAL
TSS$TRACE_OFF	RET_STATUS = TSS$TRACE_OFF
TSS$TRACE_ON	RET_STATUS = TSS$TRACE_ON
TSS$UNDECL_AFK	RET_STATUS = TSS$UNDECL_AFK & (CHANNEL, & KEY_ID)
TSS$WRITE_BRKTHRU	RET_STATUS = TSS$WRITE_BRKTHRU & (CHANNEL, & MESSAGE_TEXT, & BELL_FLAG)
TSS$WRITE_MSG_LINE	RET_STATUS = TSS$WRITE_MSG_LINE & (CHANNEL, & MESSAGE_TEXT)

Notes to Table 4-4:

* Requires the following declaration at the beginning of your BASIC program:

  EXTERNAL LONG Key_ast_routine

Table 4-5 lists the VAX TDMS synchronous calls in VAX COBOL. The calls are in alphabetical order.

Table 4-5: TDMS Synchronous Programming Calls in VAX COBOL

Call	Example
TSS$CANCEL	CALL "TSS$CANCEL" USING BY REFERENCE CHANNEL, GIVING RET-STATUS.
TSS$CLOSE	CALL "TSS$CLOSE" USING BY REFERENCE CHANNEL, BY REFERENCE CLEAR-SCREEN, GIVING RET-STATUS.
TSS$CLOSE_RLB	CALL "TSS$CLOSE_RLB" USING BY REFERENCE LIBRARY-ID, GIVING RET-STATUS.
TSS$COPY_SCREEN	CALL "TSS$COPY_SCREEN" USING BY REFERENCE CHANNEL, BY DESCRIPTOR FILE-SPEC, BY REFERENCE APPEND-FLAG, GIVING RET-STATUS.
TSS$DECL_AFK	CALL "TSS$DECL_AFK" USING BY REFERENCE CHANNEL, BY REFERENCE KEY-ID, BY REFERENCE KEY-EVENT-FLAG-NUMBER, BY REFERENCE KEY-AST-ROUTINE, BY VALUE KEY-AST-PARAMETER, GIVING RET-STATUS.
TSS$OPEN	CALL "TSS$OPEN" USING BY REFERENCE CHANNEL, BY DESCRIPTOR DEVICE, GIVING RET-STATUS.
TSS$OPEN_RLB	CALL "TSS$OPEN_RLB" USING BY DESCRIPTOR REQUEST-LIBRARY-FILE, BY REFERENCE LIBRARY-ID, GIVING RET-STATUS.

(continued on next page)

Table 4-5: TDMS Synchronous Programming Calls in VAX COBOL (Cont.)

Call	Example
TSS$READ_MSG_LINE	CALL "TSS$READ_MSG_LINE" USING BY REFERENCE CHANNEL, BY DESCRIPTOR RESPONSE-TEXT, BY DESCRIPTOR MESSAGE-PROMPT, BY REFERENCE RESPONSE-LENGTH, GIVING RET-STATUS.
TSS$REQUEST	CALL "TSS$REQUEST" USING BY REFERENCE CHANNEL, BY REFERENCE LIBRARY-ID, BY DESCRIPTOR REQUEST-NAME, BY REFERENCE RECORD-1, BY REFERENCE RECORD-2, BY REFERENCE RECORD-n, GIVING RET-STATUS.
TSS$SIGNAL	CALL "TSS$SIGNAL" GIVING RET-STATUS.
TSS$TRACE_OFF	CALL "TSS$TRACE_OFF" GIVING RET-STATUS.
TSS$TRACE_ON	CALL "TSS$TRACE_ON" GIVING RET-STATUS.
TSS$UNDECL_AFK	CALL "TSS$UNDECL_AFK" USING BY REFERENCE CHANNEL, BY REFERENCE KEY-ID, GIVING RET-STATUS.
TSS$WRITE_BRKTHRU	CALL "TSS$WRITE_BRKTHRU" USING BY REFERENCE CHANNEL, BY DESCRIPTOR MESSAGE-TEXT, BY REFERENCE BELL-FLAG, GIVING RET-STATUS.
TSS$WRITE_MSG_LINE	CALL "TSS$WRITE_MSG_LINE" USING BY REFERENCE CHANNEL, BY DESCRIPTOR MESSAGE-TEXT, GIVING RET-STATUS.

Table 4-6 lists the VAX TDMS synchronous calls in VAX FORTRAN. The calls are in alphabetical order.

Table 4-6: TDMS Synchronous Programming Calls in VAX FORTRAN

Call	Example
TSS$CANCEL	`RET_STATUS = TSS$CANCEL(%REF(CHANNEL))`
TSS$CLOSE	`RET_STATUS = TSS$CLOSE(%REF(CHANNEL),` `1 %REF(CLEAR_SCREEN))`
TSS$CLOSE_RLB	`RET_STATUS = TSS$CLOSE_RLB` `1 (%REF(LIBRARY_ID))`

TSS$COPY_SCREEN	`RET_STATUS = TSS$COPY_SCREEN`
	`1 (%REF(CHANNEL),`
	`2 %DESCR(FILE_SPEC),`
	`3 %REF(APPEND_FLAG))`

TSS$DECL_AFK	`RET_STATUS = TSS$DECL_AFK`
	`1 (%REF(CHANNEL),`
	`2 %REF(KEY_ID),`
	`4 %REF(KEY_EVENT_FLAG_NUMBER),`
	`5 %REF(KEY_AST_ROUTINE),`
	`6 KEY_AST_PARAMETER)`

TSS$OPEN	`RET_STATUS = TSS$OPEN(%REF(CHANNEL),` `1 %DESCR(DEVICE))`

TSS$OPEN_RLB	`RET_STATUS = TSS$OPEN_RLB`
	`1 (%DESCR(REQUEST_LIBRARY_FILE),`
	`2 %REF(LIBRARY_ID))`

TSS$READ_MSG_LINE	`RET_STATUS = TSS$READ_MSG_LINE`
	`1 (%REF(CHANNEL),`
	`2 %DESCR(RESPONSE_TEXT),`
	`3 %DESCR(MESSAGE_PROMPT),`
	`4 %REF(RESPONSE_LENGTH))`

TSS$REQUEST	`RET_STATUS = TSS$REQUEST`
	`1 (%REF(CHANNEL),`
	`2 %REF(LIBRARY_ID),`
	`3 %DESCR(REQUEST_NAME),`
	`4 %REF(RECORD_1),`
	`5 %REF(RECORD_2),`
	`6 %REF(RECORD_n))`

(continued on next page)

Table 4-6: TDMS Synchronous Programming Calls in VAX FORTRAN (Cont.)

Call	Example
TSS$SIGNAL	RET_STATUS = TSS$SIGNAL()
TSS$TRACE_OFF	RET_STATUS = TSS$TRACE_OFF()
TSS$TRACE_ON	RET_STATUS = TSS$TRACE_ON()
TSS$UNDECL_AFK	RET_STATUS = TSS$UNDECL_AFK (%REF(CHANNEL), 1 %REF(KEY_ID)) 2
TSS$WRITE_BRKTHRU	RET_STATUS = TSS$WRITE_BRKTHRU (%REF(CHANNEL), 1 %DESCR(MESSAGE_TEXT), 2 %REF(BELL_FLAG)) 3
TSS$WRITE_MSG_LINE	RET_STATUS = TSS$WRITE_MSG_LINE (%REF(CHANNEL), 1 %DESCR(MESSAGE_TEXT)) 2

TDMS Asynchronous Programming Calls 5

This chapter provides a complete description of all the TDMS asynchronous programming calls. Calls are listed in alphabetical order.

The discussion of each call includes:

Format	The generic description of the call. See Table 5-1 for an explanation of the notation.
Call Parameters	The description of the arguments passed.
Return Status	A general explanation of return status and a list of the codes that can be returned on that call. In addition, you see the error severity code associated with the return status. See Table 5-2 for the error severity codes and their explanation.
Completion Status	A general explanation of completion status and a list of the codes that can be returned on that call. In addition, you see the error severity code associated with the completion status. See Table 5-2 for the error severity codes and their explanation.
Notes	Additional information on using the call.
Examples	In VAX BASIC, VAX COBOL, and VAX FORTRAN.

At the end of this chapter, there are table summaries of all TDMS asynchronous programming calls by language:

- Table 5-4 in VAX BASIC
- Table 5-5 in VAX COBOL
- Table 5-6 in VAX FORTRAN

5.1 Notation Used in This Chapter

Table 5-1 gives a complete list of the data types and passing mechanisms used for TDMS programming calls.

Table 5-1: Parameter Passing Notation

Notation	Explanation
mz.r	Modifiable unspecified type, by reference
rlu.r	Read-only unsigned longword logical type, by reference
rlu .v	Read-only unsigned longword logical type, by value
rt.dx	Read-only character string, class type in descriptor
szem.r	Call without stack unwinding, procedure entry mask, by reference
wlc.v	Write-only longword return status, by value
wlu.r	Write-only unsigned longword logical type, by reference
wt.dx	Write-only character string, class type in descriptor
wwu.r	Write-only unsigned word logical type, by reference

All TDMS programming calls return a standard VAX/VMS status code. Each status code has an associated severity level. Table 5-2 explains the possible severity levels of the status codes that TDMS programming calls return.

Table 5-2: Error Severity Codes for Return Status and Completion Status

Error Severity	Explanation
S	Successful completion
I	Information only
F	Indicates a Severe error; execution of the program does not continue, and the procedure does not produce any output

5.2 TSS$CLOSE_A Call

Closes the channel to the terminal and optionally clears the screen. Initiates this operation and then returns control immediately to the application program.

Format

```
ret-status.wlc.v = TSS$CLOSE_A(channel.rlu.r
                               ,[rsb.wlu.r]
                               { | ,efn.rlu.r      | }
                               { | ,astadr.szem.r  | }
                               { |   ,[astprm.rlu.v] | }
                               [,clear-screen.rlu.r])
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN or TSS$OPEN_A call; you pass this parameter by reference.

rsb

The address of the longword to receive the completion status for the call; you pass this optional parameter by reference. If the parameter is not present, it is passed as a 0.

efn

The address of the longword containing the number of the event flag set at call completion; you pass this parameter by reference. If the parameter is not present, it is passed as 0 and TDMS does not set an event flag for this call.

Either the event flag parameter or the AST routine parameter must be present for the call.

astadr

The address of a routine in the application program; you pass this parameter by reference. If the parameter is present, an AST is declared for this service routine at call completion.

Either the event flag parameter or the AST routine parameter must be present for the call.

astprm

The longword containing the AST parameter to be passed to the AST routine upon call completion; you pass this optional parameter by value.

If the AST parameter is not present, and a service routine is, TDMS will pass an AST parameter of 0 to the service routine.

TDMS treats this parameter as a value: you can pass any type of parameter you would like your AST routine to receive, including addresses (parameters by reference).

clear-screen

The address of the longword specifying whether or not the screen is cleared; you pass this parameter by reference. If the value of this parameter is 1, then the screen is cleared. The clear-screen parameter is optional, and if it is omitted or if the parameter is 0, the screen is not cleared.

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$_BADFLAGS

Flag parameter has invalid value (F)

TSS$_BUGCHECK

Fatal internal software error (F)

TSS$_CANINPROG

Cancel in progress on channel (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_IOINPROG

I/O in progress on channel (F)

TSS$_NORMAL

Normal successful completion (S)

Notes

You should close the TDMS channel when you are finished using TDMS for input/output on that terminal. As a result:

- TDMS resources associated with that terminal are released
- Screen video attributes are reset
- The cursor is returned to the bottom left hand corner of the terminal screen
- The keypad is reset to Numeric mode

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
              .
              .
              .

Return_status = TSS$CLOSE_A(Channel, &
                            Return_status_block, &
                            Event_flag_number, &
                            Ast_routine, &
                            Ast_parameter BY VALUE, &
                            Clear_screen)
```

COBOL

```
CALL "TSS$CLOSE_A"
     USING BY REFERENCE Channel,
           BY REFERENCE Return-status-block,
           BY REFERENCE Event-flag-number,
           BY REFERENCE Ast-routine,
           BY VALUE Ast-parameter,
           BY REFERENCE Clear-screen,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$CLOSE_A(%REF(Channel),
1                            %REF(Return_status_block),
2                            %REF(Event_flag_number),
3                            %REF(Ast_routine),
4                            Ast_parameter,
5                            %REF(Clear_screen))
```

5.3 TSS$COPY_SCREEN_A Call

Copies the current contents of the screen to the file specified or the file defined by the logical TSS$HARDCOPY. Initiates this operation and then returns control immediately to the application program.

Format

```
ret-status.wlc.v = TSS$COPY_SCREEN_A(channel.rlu.r
                                     ,[rsb.wlu.r]
                                     {  |,efn.rlu.r          | }
                                     {  |                    | }
                                     {  |,astadr.szem.r       | }
                                     {  |  ,[astprm.rlu.v]    | }
                                     ,[file-spec.rt.dx]
                                     [,append-flag.rlu.r])
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN or TSS$OPEN_A call; you pass this parameter by reference.

rsb

The address of the longword to receive the completion status for the call; you pass this optional parameter by reference. If the parameter is not present, it is passed as a 0.

efn

The address of the longword containing the number of the event flag set at call completion; you pass this parameter by reference. If the parameter is not present, it is passed as 0 and TDMS does not set an event flag for this call.

Either the event flag parameter or the AST routine parameter must be present for the call.

astadr

The address of a routine in the application program; you pass this parameter by reference. If the parameter is present, an AST is declared for this service routine at call completion.

Either the event flag parameter or the AST routine parameter must be present for the call.

astprm

The longword containing the AST parameter to be passed to the AST routine upon call completion; you pass this optional parameter by value.

If the AST parameter is not present, and a service routine is, TDMS passes an AST parameter of 0 to the service routine.

TDMS treats this parameter as a value: you can pass any type of parameter you would like your AST routine to receive, including addresses (parameters by reference).

file-spec

The address of a string descriptor pointing to the VMS file specification to which the contents of the screen will be directed. You pass this optional parameter by descriptor.

If this parameter is not specified, the value of the logical TSS$HARDCOPY is used to determine where the contents of the screen are directed.

If this parameter is not specified and the logical TSS$HARDCOPY is not defined, this call does nothing. The return status is TSS$_NOOUTFILE, an informational status.

append-flag

The address of a longword specifying whether to create a new version of the file or to append the copy of the screen to the latest version of the file. You pass this optional parameter by reference. If present, this parameter must have a value of either 0 or 1.

A new version of the output file will be created if:

- The parameter is not present
- The flag has a value of 0
- There is no existing version of the file

Otherwise, the current contents of the screen will be appended to the latest version of an existing VMS file.

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$_BADFLAGS

Flag parameter has invalid value (F)

TSS$_BUGCHECK

Fatal internal software error (F)

TSS$_CANCEL

Call canceled by TSS$CANCEL (F)

TSS$_CANINPROG

Cancel in progress on channel (F)

TSS$_COPYOUTERR

TSS$COPY_SCREEN output error (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_IOINPROG

I/O in progress on channel (F)

TSS$_NOOUTFILE

No output file definition found (I)

TSS$_NORMAL

Normal successful completion (S)

Notes

TSS$COPY_SCREEN_A supplies a callable PF4 (hardcopy) function to the program. You cannot issue a TSS$COPY_SCREEN_A call while input or output is active on the specified channel, such as with an outstanding TSS$REQUEST or TSS$WRITE_MSG_LINE call.

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.

TSS$COPY_SCREEN_A

Examples

BASIC

```
EXTERNAL LONG Ast_routine
            .
            .
            .

Return_status = TSS$COPY_SCREEN_A(Channel, &
                                  Return_status_block, &
                                  Event_flag_number, &
                                  Ast_routine, &
                                  Ast_parameter BY VALUE, &
                                  File_spec, &
                                  Append_flag)
```

COBOL

```
CALL "TSS$COPY_SCREEN_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 File-spec,
           BY REFERENCE Append-flag,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$COPY_SCREEN_A(%REF(Channel),
1                                  %REF(Return_status_block),
2                                  %REF(Event_flag_number),
3                                  %REF(Ast_routine),
4                                  Ast_parameter,
5                                  %DESCR(File_spec),
6                                  %REF(Append_flag))
```

5.4 TSS$DECL_AFK_A Call

Enables the operator's use of an application function key (AFK) by specifying the AFK and associating it with a service routine and/or event flag. Initiates this operation and then returns control immediately to the application program.

Format

```
ret-status.wlc.v = TSS$DECL_AFK_A(channel.rlu.r
                                  ,[rsb.wlu.r]
                                  { ,efn.rlu.r                  }
                                  { ,astadr.szem.r              }
                                  {   ,[astprm.rlu.v]           }
                                  ,key-id.rlu.r
                                  { ,key-efn.rlu.r              }
                                  { ,key-astadr.szem.r          }
                                  {   [,key-astprm.rlu.v]       } )
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN or TSS$OPEN_A call; you pass this parameter by reference.

rsb

The address of the longword to receive the completion status for the call; you pass this optional parameter by reference. If the parameter is not present, it is passed as a 0.

efn

The address of the longword containing the number of the event flag set at call completion; you pass this parameter by reference. If the parameter is not present, it is passed as 0 and TDMS does not set an event flag for this call.

Either the event flag parameter or the AST routine parameter must be present for the call.

astadr

The address of a routine in the application program; you pass this parameter by reference. If the parameter is present, an AST is declared for this service routine at call completion.

Either the event flag parameter or the AST routine parameter must be present for the call.

astprm

The longword containing the AST parameter to be passed to the AST routine upon call completion; you pass this optional parameter by value.

If the AST parameter is not present and a service routine is, TDMS passes an AST parameter of 0 to the service routine.

TDMS treats this parameter as a value: you can pass any type of parameter you would like your AST routine to receive, including addresses (parameters by reference).

key-id

The address of a longword containing a code that represents the AFK; you pass this parameter by reference. When the operator presses the key represented by this key-id parameter, the application program is notified.

Because the key-id parameter is a code, each call to TSS$DECL_AFK_A can activate only one key. Table 5-3 is a list of TDMS application function keys.

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 may use the AST service routine with or without an AST parameter. You may also use the AST service routine with an event flag.

key-astprm

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

TDMS treats this parameter as a value: you can pass any type of parameter you would like your AST routine to receive, including addresses (parameters by reference).

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)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVKEYID

Invalid key id (F)

TSS$_NORMAL

Normal successful completion (S)

Notes

Application function keys (AFKs) provide exception notification services for terminal-related events. During execution of a TDMS application, an 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 application program.

You can enable and disable the operator's use of AFKs by issuing TSS$DECL_AFK_A and TSS$UNDECL_AFK_A calls in the application program. The TSS$DECL_AFK_A call specifies the AFK by the key-id parameter and associates that key with a service routine and/or event flag. After the program has made a TSS$DECL_AFK_A 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

When an AFK is pressed, TDMS does one of the following:

- Sets the event flag specified for this AFK (if any)
- Calls the service routine (if any) at AST level and passes it:
  - The AST parameter
  - The TDMS channel on which it was pressed
  - The key-id parameter of the AFK that was pressed
- Both sets the event flag and calls the service routine

  In this case, TDMS sets the event flag before it calls the service routine.

TDMS calls the service routine whether or not there is an outstanding request, as soon as the VMS terminal driver processes the keystroke.

All the control keys except CTRL/Q and CTRL/S may be defined as AFKs.

Note that some control keys (for example, CTRL/H or BACK SPACE, CTRL/I or TAB, CTRL/J or LINE FEED, and CTRL/M or RETURN) may also be defined as AFKs. You should be careful when designing functions for AFKs that already have a special meaning for TDMS; otherwise, unexpected results might be generated. For example, if you define CTRL/M (RETURN) as an AFK, you must provide another way of terminating the request.

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
EXTERNAL LONG Key_ast_routine
             .
             .
             .

Return_status = TSS$DECL_AFK_A(Channel, &
                               Return_status_block, &
                               Event_flag_number, &
                               Ast_routine, &
                               Ast_parameter BY VALUE, &
                               Key_id, &
                               Key_event_flag_number, &
                               Key_ast_routine,  &
                               Key_ast_parameter BY VALUE)
```

COBOL

```
CALL "TSS$DECL_AFK_A"
     USING BY REFERENCE Channel,
           BY REFERENCE Return-status-block,
           BY REFERENCE Event-flag-number,
           BY REFERENCE Ast-routine,
           BY VALUE Ast-parameter,
           BY REFERENCE Key-id,
           BY REFERENCE Key-event-flag-number,
           BY REFERENCE Key-ast-routine,
           BY VALUE Key-ast-parameter,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$DECL_AFK_A(%REF(Channel),
1                               %REF(Return_status_block),
2                               %REF(Event_flag_number),
3                               %REF(Ast_routine),
4                               Ast_parameter,
5                               %REF(Key_id),
6                               %REF(Key_event_flag_number),
7                               %REF(Key_ast_routine),
8                               Key_ast_parameter)
```

5.5 TSS$OPEN_A Call

Opens a channel to a terminal for input and output. Initiates this operation and then returns control immediately to the application program.

Format

```
ret-status.wlc.v = TSS$OPEN_A(channel.wlu.r
                              ,[rsb.wlu.r]

                              { | ,efn.rlu.r                | }
                              { | ,astadr.szem.r            | }
                              { |   ,[astprm.rlu.v]         | }

                              [,device.rt.dx])
```

Call Parameters

channel

The address of a longword to receive the TDMS channel number; you pass this parameter by reference.

rsb

The address of the longword to receive the completion status for the call; you pass this optional parameter by reference. If the parameter is not present, it is passed as a 0.

efn

The address of the longword containing the number of the event flag set at call completion; you pass this parameter by reference. If the parameter is not present, it is passed as 0 and TDMS does not set an event flag for this call.

Either the event flag parameter or the AST routine parameter must be present for the call.

astadr

The address of a routine in the application program; you pass this parameter by reference. If the parameter is present, an AST is declared for this service routine at call completion.

Either the event flag parameter or the AST routine parameter must be present for the call.

astprm

The longword containing the AST parameter to be passed to the AST routine upon call completion; you pass this optional parameter by value.

If the AST parameter is not present, and a service routine is, TDMS passes an AST parameter of 0 to the service routine.

TDMS treats this parameter as a value: you can pass any type of parameter you would like your AST routine to receive, including addresses (parameters by reference).

device

The address of a string descriptor pointing to the device name of the terminal; you pass this optional parameter by descriptor. This can be a logical name or a physical device specification. This parameter is optional; the default is SYS$INPUT.

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$_ERROPNDEV

Error opening device (F)

TSS$_ILLDEVCHAR

Illegal device characteristics (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_NORMAL

Normal successful completion (S)

Notes

TDMS assigns a unique channel number to identify the terminal. You use this channel number as a parameter in other TDMS calls to identify what terminal to use for the operation.

Input and output must be performed on the same terminal; therefore, the input/output streams cannot be split between two terminals. For example, if you open a channel to SYS$INPUT and you redefine SYS$OUTPUT as some other device, both input and output still occur on SYS$INPUT.

The channel number returned is not the VMS channel number returned by the SYS$ASSIGN service. Input/output calls to other systems (such as VAX RMS or $QIO) *should not* be issued to the terminal. If you try to issue calls to other systems, you might generate unexpected results.

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
            .
            .
            .

Return_status = TSS$OPEN_A(Channel,
                           Return_status_block, &
                           Event_flag_number, &
                           Ast_routine,  &
                           Ast_parameter BY VALUE, &
                           Device_name)
```

COBOL

```
CALL "TSS$OPEN_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 Device-name,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$OPEN_A(%REF(Channel),
1                           %REF(Return_status_block),
2                           %REF(Event_flag_number),
3                           %REF(Ast_routine),
4                           Ast_parameter,
5                           %DESCR(Device_name))
```

5.6 TSS$READ_MSG_LINE_A Call

Writes a prompt on the screen on the reserved message line (optionally), and reads a line of input from the operator. Initiates this operation and then returns control immediately to the application program.

Format

```
ret-status.wlc.v = TSS$READ_MSG_LINE_A(channel.rlu.r
                                       ,[rsb.wlu.r]
                                       
                                       { | ,efn.rlu.r                | }
                                       { |                           | }
                                       { | ,astadr.szem.r             | }
                                       { |   ,[astprm.rlu.v]          | }
                                       
                                       ,response-text.wt.dx
                                       ,[message-prompt.rt.dx]
                                       [,response-length.wwu.r])
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN or TSS$OPEN_A call; you pass this parameter by reference.

rsb

The address of the longword to receive the completion status for the call; you pass this optional parameter by reference. If the parameter is not present, it is passed as a 0.

efn

The address of the longword containing the number of the event flag set at call completion; you pass this parameter by reference. If the parameter is not present, it is passed as 0 and TDMS does not set an event flag for this call.

Either the event flag parameter or the AST routine parameter must be present for the call.

astadr

The address of a routine in the application program; you pass this parameter by reference. If the parameter is present, an AST is declared for this service routine at call completion.

Either the event flag parameter or the AST routine parameter must be present for the call.

astprm

The longword containing the AST parameter to be passed to the AST routine upon call completion; you pass this optional parameter by value.

If the AST parameter is not present, and a service routine is, TDMS passes an AST parameter of 0 to the service routine.

TDMS treats this parameter as a value: you can pass any type of parameter you would like your AST routine to receive, including addresses (parameters by reference).

response-text

The address of a static or dynamic string descriptor pointing to the buffer to receive the text entered by the operator; you pass this optional parameter by descriptor.

message-prompt

The address of a string descriptor pointing to the prompt you want to display on the message line; you pass this optional parameter by descriptor.

response-length

The address of a word to receive the length of the message line read into the buffer; you pass this optional parameter by reference.

This parameter is useful for setting the length of the response-text parameter so that it can be compared with other strings within the application program. If you use a static string descriptor for the response-text parameter and omit the response-length parameter, you cannot tell how long the response text is. A comparison might fail because the two strings are unequal in length.

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$_CANCEL

Call canceled by TSS$CANCEL (F)

TSS$_CANINPROG

Cancel in progress on channel (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_INVTXTLEN

Invalid text length (F)

TSS$_IOINPROG

I/O in progress on channel (F)

TSS$_NONPRICHA

Field contains non-printable characters (F)

TSS$_NORMAL

Normal successful completion (S)

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.

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))
```

5.7 TSS$REQUEST_A Call

Executes the request named in the call. Initiates this operation and then returns control immediately to the application program.

Format

```
ret-status.wlc.v = TSS$REQUEST_A(channel.rlu.r
                                 ,[rsb.wlu.r]

                                 ( | ,efn.rlu.r                | )
                                 { |                           | }
                                 { | ,astadr.szem.r            | }
                                 ( |   ,[astprm.rlu.v]         | )

                                 ,library-id.rlu.r
                                 ,request-name.rt.dx
                                 [,record1.mz.r
                                 [,record2.mz.r
                                    .
                                    .
                                    .
                                 [,recordn.mz.r]]])
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN or TSS$OPEN_A call; you pass this parameter by reference.

rsb

The address of the longword to receive the completion status for the call; you pass this optional parameter by reference. If the parameter is not present, it is passed as a 0.

efn

The address of the longword containing the number of the event flag set at call completion; you pass this parameter by reference. If the parameter is not present, it is passed as 0 and TDMS does not set an event flag for this call.

Either the event flag parameter or the AST routine parameter must be present for the call.

astadr

The address of a routine in the application program; you pass this parameter by reference. If the parameter is present, an AST is declared for this service routine at call completion.

Either the event flag parameter or the AST routine parameter must be present for the call.

astprm

The longword containing the AST parameter to be passed to the AST routine upon call completion; you pass this optional parameter by value.

If the AST parameter is not present, and a service routine is, TDMS passes an AST parameter of 0 to the service routine.

TDMS treats this parameter as a value: you can pass any type of parameter you would like your AST routine to receive, including addresses (parameters by reference).

library-id

The address of the longword containing the unique number that identifies the library containing the desired request; you pass this parameter by reference. It must be the same number that was assigned by TSS$OPEN_RLB.

request-name

The address of a string descriptor pointing to the name of the request to use; you pass this parameter by descriptor.

record(s)

The address of the record(s) the request refers to for mapping data between the form and the program record; you pass this optional parameter by reference. If you specify more than one record, the order of the records must be the same as in the RECORD IS instruction(s) in the request.

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$_CANCEL

Call canceled by TSS$CANCEL (F)

TSS$_CANINPROG

Cancel in progress on channel (F)

TSS$_CONVERR

Data type conversion error (F)

TSS$_DISFORERR

Form display failed (F)

TSS$_ILLDEVCHAR

Illegal device characteristics (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_MULTFORM

Multiple active forms are illegal (F)

TSS$_NORMAL

Normal successful completion (S)

TSS$_PRKCHECK

Request was terminated by a check PRK (I)

TSS$_PRKNOCHECK

Request was terminated by a nocheck PRK (I)

TSS$_REQMAJVERMIS

Request binary version mismatch on major id (F)

TSS$_REQMINVERMIS

Request binary version mismatch on minor id (F)

TSS$_REQNOTFOU

Request definition not found (F)

Notes

The request must be in a currently open request library file.

If your programming language supports the extraction of records from the CDD, record structure should not be a problem because you can use the same record definition in your program that was used to build the request library file. Note that if you use CDDL or DATATRIEVE to define your records, you must pass a variable name to TSS$REQUEST_A that corresponds to the top level structure name in the record definition. With languages that do not support the CDD, you must be careful to define your records so that they are compatible with your programming language. Otherwise you will generate unexpected results.

The order of the records passed to TDMS must match the order of the records in the RECORD IS instructions in the request.

Because you pass records by reference, TDMS has no way of validating that you are passing the correct records.

The order in which instructions are specified in the request has no effect on the order in which TDMS executes the instructions. TDMS executes request instructions in the following order (if they are present in the request):

1. Control field evaluation
2. Output mappings from the program record to the form, including data type conversion to text format for display, default field instructions, and video instructions
3. Display of the form
4. Input mappings from the terminal to the program, including data type conversion from text to the data type of the record field(s)
5. Return operations

The following form definition features cannot be used on a VT52 terminal:

- 132-column screen
- Scrolled region

You generate a run-time error *only when* you try to display a form definition containing one of those features. For example, if you run an application on a VT52 and use form definitions without those features for the first five request calls, all of those calls will be successful. However, if the sixth request call contains a form definition with one of those features, it fails with the message:

%TSS-F-ILLDEVCHAR, illegal device characteristics

You should either change the form definition or run the application on a VT100-compatible terminal.

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 causes 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
              .
              .
              .

Return_status = TSS$REQUEST_A(Channel, &
                              Return_status_block, &
                              Event_flag_number, &
                              Ast_routine, &
                              Ast_parameter BY VALUE, &
                              Library_id, &
                              Request_name, &
                              Record_1, &
                              Record_2, &
                              Record_n)
```

COBOL

```
CALL "TSS$REQUEST_A"
     USING BY REFERENCE Channel,
           BY REFERENCE Return-status-block,
           BY REFERENCE Event-flag-number,
           BY REFERENCE Ast-routine,
           BY VALUE Ast-parameter,
           BY REFERENCE Library-id,
           BY DESCRIPTOR Request-name,
           BY REFERENCE Record-1,
           BY REFERENCE Record-2,
           BY REFERENCE Record-n,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$REQUEST_A(%REF(Channel),
1                              %REF(Return_status_block),
2                              %REF(Event_flag_number),
3                              %REF(Ast_routine),
4                              Ast_parameter,
5                              %REF(Library_id),
6                              %DESCR(Request_name),
7                              %REF(Record_1),
8                              %REF(Record_2),
9                              %REF(Record_n))
```

5.8 TSS$UNDECL_AFK_A Call

Disables an application function key (AFK) and its associated service routine and/or event flag. Initiates this operation and then returns control immediately to the application program.

Format

```
ret-status.wlc.v = TSS$UNDECL_AFK_A(channel.rlu.r
                                    [rsb.wlu.r]
                                    {[,efn.rlu.r]
                                    {[,astadr.szem.r]
                                    {  [,astprm.rlu.v]]
                                    ,key-id.rlu.r)
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN or TSS$OPEN_A call; you pass this parameter by reference.

rsb

The address of the longword to receive the completion status for the call; you pass this optional parameter by reference. If the parameter is not present, it is passed as a 0.

efn

The address of the longword containing the number of the event flag set at call completion; you pass this parameter by reference. If the parameter is not present, it is passed as 0 and TDMS does not set an event flag for this call.

Either the event flag parameter or the AST routine parameter must be present for the call.

astadr

The address of a routine in the application program; you pass this parameter by reference. If the parameter is present, an AST is declared for this service routine at call completion.

Either the event flag parameter or the AST routine parameter must be present for the call.

astprm

The longword containing the AST parameter to be passed to the AST routine upon call completion; you pass this optional parameter by value.

If the AST parameter is not present, and a service routine is, TDMS passes an AST parameter of 0 to the service routine.

TDMS treats this parameter as a value: you can pass any type of parameter you would like your AST routine to receive, including addresses (parameters by reference).

key-id

The address of a longword containing the code representing the AFK that is no longer needed by the application program. You pass this parameter by reference.

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)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVKEYID

Invalid key id (F)

TSS$_NORMAL

Normal successful completion (S)

Notes

You use this call to deactivate asynchronous notification of an AFK. After the TSS$UNDECL_AFK_A call is issued, TDMS no longer calls the service routine specified in the matching TSS$DECL_AFK_A call. That is, after the TSS$UNDECL_AFK_A call, the program no longer is notified when the operator presses the key.

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 causes 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
            .
            .
            .

Return_status = TSS$UNDECL_AFK_A(Channel), &
                                 Return_status_block, &
                                 Event_flag_number, &
                                 Ast_routine, &
                                 Ast_parameter BY VALUE, &
                                 Key_id)
```

COBOL

```
CALL "TSS$UNDECL_AFK_A"
     USING BY REFERENCE Channel,
           BY REFERENCE Return-status-block,
           BY REFERENCE Event-flag-number,
           BY REFERENCE Ast-routine,
           BY VALUE Ast-parameter,
           BY REFERENCE Key-id,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$UNDECL_AFK_A(%REF(Channel),
1                                 %REF(Return_status_block),
2                                 %REF(Event_flag_number),
3                                 %REF(Ast_routine),
4                                 Ast_parameter,
5                                 %REF(Key_id))
```

5.9 TSS$WRITE_BRKTHRU_A Call

Writes a message to the reserved message line on the screen, interrupting the current request or message line operation in order to do so. Initiates this operation and then returns control immediately to the application program.

Format

```
ret-status.wlc.v = TSS$WRITE_BRKTHRU_A(channel.rlu.r
                                       ,[rsb.wlu.r]
                                       { | ,efn.rlu.r                | }
                                       { | ,astadr.szem.r            | }
                                       { |   ,[astprm.rlu.v]         | }
                                       ,message-text.rt.dx
                                       [,bell-flag.rlu.r])
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN or TSS$OPEN_A call; you pass this parameter by reference.

rsb

The address of the longword to receive the completion status for the call; you pass this optional parameter by reference. If the parameter is not present, it is passed as a 0.

efn

The address of the longword containing the number of the event flag set at call completion; you pass this parameter by reference. If the parameter is not present, it is passed as 0 and TDMS does not set an event flag for this call.

Either the event flag parameter or the AST routine parameter must be present for the call.

astadr

The address of a routine in the application program; you pass this parameter by reference. If the parameter is present, an AST is declared for this service routine at call completion.

Either the event flag parameter or the AST routine parameter must be present for the call.

astprm

The longword containing the AST parameter to be passed to the AST routine upon call completion; you pass this optional parameter by value.

If the AST parameter is not present, and a service routine is, TDMS passes an AST parameter of 0 to the service routine.

TDMS treats this parameter as a value: you can pass any type of parameter you would like your AST routine to receive, including addresses (parameters by reference).

message-text

The address of a string descriptor pointing to the text to be displayed on the message line; you pass this parameter by descriptor.

bell-flag

The address of a longword containing a flag for the terminal bell; you pass this optional parameter by reference. If this parameter is set to 1, the flag causes the terminal bell to ring when any message text is displayed. If you do not pass this parameter or if this parameter has a value of 0, TDMS does not ring the bell.

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$_BADFLAGS

Flag parameter has invalid value (F)

TSS$_BUGCHECK

Fatal internal software error (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_NONPRICHA

Field contains non-printable characters (F)

TSS$_NORMAL

Normal successful completion (S)

Notes

The message text for this call is truncated if its length is greater than the current terminal line size.

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 causes 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
             .
             .
             .

Return_status = TSS$WRITE_BRKTHRU_A(Channel, &
                                    Return_status_block, &
                                    Event_flag_number, &
                                    Ast_routine, &
                                    Ast_parameter BY VALUE, &
                                    Message_text, &
                                    Bell_flag)
```

COBOL

```
CALL "TSS$WRITE_BRKTHRU_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 Message-text,
           BY REFERENCE Bell-flag,
     GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$WRITE_BRKTHRU_A(%REF(Channel),
1                                    %REF(Return_status_block),
2                                    %REF(Event_flag_number),
3                                    %REF(Ast_routine),
4                                    Ast_parameter,
5                                    %DESCR(Message_text),
6                                    %REF(Bell_flag))
```

5.10 TSS$WRITE_MSG_LINE_A Call

Writes a message to the reserved message line on the screen. Initiates this operation and then returns control immediately to the application program.

Format

```
ret-status.wlc.v = TSS$WRITE_MSG_LINE_A(channel.rlu.r
                                        ,[rsb.wlu.r]
                                        { | ,efn.rlu.r                | }
                                        { | ,astadr.szem.r            | }
                                        { |   ,[astprm.rlu.v]         | }
                                        ,message-text.rt.dx)
```

Call Parameters

channel

The address of the longword containing the unique TDMS channel number assigned on the TSS$OPEN or TSS$OPEN_A call; you pass this parameter by reference.

rsb

The address of the longword to receive the completion status for the call; you pass this optional parameter by reference. If the parameter is not present, it is passed as a 0.

efn

The address of the longword containing the number of the event flag set at call completion; you pass this parameter by reference. If the parameter is not present, it is passed as 0 and TDMS does not set an event flag for this call.

Either the event flag parameter or the AST routine parameter must be present for the call.

astadr

The address of a routine in the application program; you pass this parameter by reference. If the parameter is present, an AST is declared for this service routine at call completion.

Either the event flag parameter or the AST routine parameter must be present for the call.

astprm

The longword containing the AST parameter to be passed to the AST routine upon call completion; you pass this optional parameter by value.

If the AST parameter is not present, and a service routine is, TDMS passes an AST parameter of 0 to the service routine.

TDMS treats this parameter as a value: you can pass any type of parameter you would like your AST routine to receive, including addresses (parameters by reference).

message-text

The address of a string descriptor pointing to the text to be displayed on the message line; you pass this parameter by descriptor.

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$_CANCEL

Call canceled by TSS$CANCEL (F)

TSS$_CANINPROG

Cancel in progress on channel (F)

TSS$_INSVIRMEM

Insufficient virtual memory (F)

TSS$_INVARG

Invalid arguments (F)

TSS$_INVCHN

Invalid channel (F)

TSS$_INVDSC

Invalid descriptor (F)

TSS$_INVTXTLEN

Invalid text length (F)

TSS$_NONPRICHA

Field contains non-printable characters (F)

TSS$_NORMAL

Normal successful completion (S)

Notes

TDMS clears the reserved message line before it displays the message on the screen. You can use this call to let the operator know about the status of AFKs.

TDMS uses the message line to display error messages; therefore, it is uncertain how long the line will be displayed. The cases when the message line is cleared are:

- If the operator makes an error on input
- If the operator exits the current input field
- When the screen is cleared

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
              .
              .
              .

Return_status = TSS$WRITE_MSG_LINE_A(Channel, &
                                     Return_status_block, &
                                     Event_flag_number, &
                                     Ast_routine, &
                                     Ast_parameter BY VALUE, &
                                     Message_text)
```

COBOL

```
CALL "TSS$WRITE_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 Message-text,
     GIVING Return-status.
```

TSS$WRITE_MSG_LINE_A

FORTRAN

```
 Return_status = TSS$WRITE_MSG_LINE_A(%REF(Channel),
1                                     %REF(Return_status_block),
2                                     %REF(Event_flag_number),
3                                     %REF(Ast_routine),
4                                     Ast_parameter,
5                                     %DESCR(Message_text))
```

Table 5-4 lists the VAX TDMS asynchronous programming calls in VAX BASIC. The calls are in alphabetical order.

Table 5-4: TDMS Asynchronous Programming Calls in VAX BASIC

Call	Example
TSS$CLOSE_A +	RET_STATUS = TSS$CLOSE_A & (CHANNEL, & RETURN_STATUS_BLOCK, & EVENT_FLAG_NUMBER, & AST_ROUTINE, & AST_PARAMETER BY VALUE, & CLEAR_SCREEN)
TSS$COPY_SCREEN_A +	RET_STATUS = TSS$COPY_SCREEN_A & (CHANNEL, & RETURN_STATUS_BLOCK, & EVENT_FLAG_NUMBER, & AST_ROUTINE, & AST_PARAMETER BY VALUE, & FILE_SPEC, & APPEND_FLAG)
TSS$DECL_AFK_A * +	RET_STATUS = TSS$DECL_AFK_A & (CHANNEL, & RETURN_STATUS_BLOCK, & EVENT_FLAG_NUMBER, & AST_ROUTINE, & AST_PARAMETER BY VALUE, & KEY_ID, & KEY_EVENT_FLAG_NUMBER, & KEY_AST_ROUTINE, & KEY_AST_PARAMETER BY VALUE)
TSS$OPEN_A +	RET_STATUS = TSS$OPEN_A & (CHANNEL, & RETURN_STATUS_BLOCK, & EVENT_FLAG_NUMBER, & AST_ROUTINE, & AST_PARAMETER BY VALUE, & DEVICE)

(continued on next page)

Table 5-4: TDMS Asynchronous Programming Calls in VAX BASIC (Cont.)

Call	Example
TSS$READ_MSG_LINE_A +	RET_STATUS = TSS$READ_MSG_LINE_A & (CHANNEL, & RETURN_STATUS_BLOCK, & EVENT_FLAG_NUMBER, & AST_ROUTINE, & AST_PARAMETER BY VALUE, & RESPONSE_TEXT, & MESSAGE_PROMPT, & RESPONSE_LENGTH)
TSS$REQUEST_A +	RET_STATUS = TSS$REQUEST_A & (CHANNEL, & RETURN_STATUS_BLOCK, & EVENT_FLAG_NUMBER, & AST_ROUTINE, & AST_PARAMETER BY VALUE, & LIBRARY_ID, & REQUEST_NAME, & RECORD_1, & RECORD_2, & RECORD_n)
TSS$UNDECL_AFK_A +	RET_STATUS = TSS$UNDECL_AFK_A & (CHANNEL, & RETURN_STATUS_BLOCK, & EVENT_FLAG_NUMBER, & AST_ROUTINE, & AST_PARAMETER BY VALUE, & KEY_ID)
TSS$WRITE_BRKTHRU_A +	RET_STATUS = TSS$WRITE_BRKTHRU_A & (CHANNEL, & RETURN_STATUS_BLOCK, & EVENT_FLAG_NUMBER, & AST_ROUTINE, & AST_PARAMETER BY VALUE, & MESSAGE_TEXT, & BELL_FLAG)
TSS$WRITE_MSG_LINE_A +	RET_STATUS = TSS$WRITE_MSG_LINE_A & (CHANNEL, & RETURN_STATUS_BLOCK, & EVENT_FLAG_NUMBER, & AST_ROUTINE, & AST_PARAMETER BY VALUE, & MESSAGE_TEXT)

Notes to Table 5-4:

* Requires the following declaration at the beginning of your BASIC program:

  EXTERNAL LONG Key_ast_routine

+ Requires the following declaration at the beginning of your BASIC program:

  EXTERNAL LONG Ast_routine

Table 5-5 lists the VAX TDMS asynchronous programming calls in VAX COBOL. The calls are in alphabetical order.

Table 5-5: TDMS Asynchronous Programming Calls in VAX COBOL

Call	Example
TSS$CLOSE_A	CALL "TSS$CLOSE_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY REFERENCE CLEAR-SCREEN, GIVING RET-STATUS.
TSS$COPY_SCREEN_A	CALL "TSS$COPY_SCREEN_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 FILE-SPEC, BY REFERENCE APPEND-FLAG, GIVING RET-STATUS.
TSS$DECL_AFK_A	CALL "TSS$DECL_AFK_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY REFERENCE KEY-ID, BY REFERENCE KEY-EVENT-FLAG-NUMBER, BY REFERENCE KEY-AST-ROUTINE, BY VALUE KEY-AST-PARAMETER, GIVING RET-STATUS.

(continued on next page)

Table 5-5: TDMS Asynchronous Programming Calls in VAX COBOL (Cont.)

Call	Example
TSS$OPEN_A	CALL "TSS$OPEN_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 DEVICE, GIVING RET-STATUS.
TSS$READ_MSG_LINE_A	CALL "TSS$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 RET-STATUS.
TSS$REQUEST_A	CALL TSS$REQUEST_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY REFERENCE LIBRARY-ID, BY DESCRIPTOR REQUEST-NAME, BY REFERENCE RECORD-1, BY REFERENCE RECORD-2, BY REFERENCE RECORD-n, GIVING RET-STATUS.
TSS$UNDECL_AFK_A	CALL "TSS$UNDECL_AFK_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY REFERENCE KEY-ID, GIVING RET-STATUS.

(continued on next page)

Table 5-5: TDMS Asynchronous Programming Calls in VAX COBOL (Cont.)

Call	Example
TSS$WRITE_BRKTHRU_A	CALL "TSS$WRITE_BRKTHRU_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 MESSAGE-TEXT, BY REFERENCE BELL-FLAG, GIVING RET-STATUS.
TSS$WRITE_MSG_LINE_A	CALL "TSS$WRITE_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 MESSAGE-TEXT, GIVING RET-STATUS.

Table 5-6 lists the VAX TDMS asynchronous programming calls in VAX FORTRAN. The calls are in alphabetical order.

Table 5-6: TDMS Asynchronous Programming Calls in VAX FORTRAN

Call	Example
TSS$CLOSE_A	RET_STATUS = TSS$CLOSE_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %REF(CLEAR_SCREEN))
TSS$COPY_SCREEN_A	RET_STATUS = TSS$COPY_SCREEN_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %DESCR(FILE_SPEC), 7 %REF(APPEND_FLAG))

(continued on next page)

Table 5-6: TDMS Asynchronous Programming Calls in VAX FORTRAN (Cont.)

Call	Example
TSS$DECL_AFK_A	RET_STATUS = TSS$DECL_AFK_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %REF(KEY_ID), 7 %REF(KEY_EVENT_FLAG_NUMBER), 8 %REF(KEY_AST_ROUTINE), 9 KEY_AST_PARAMETER)
TSS$OPEN_A	RET_STATUS = TSS$OPEN_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %DESCR(DEVICE))
TSS$READ_MSG_LINE_A	RET_STATUS = TSS$READ_MSG_LINE_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %DESCR(RESPONSE_TEXT), 7 %DESCR(MESSAGE_PROMPT), 8 %REF(RESPONSE_LENGTH))
TSS$REQUEST_A	RET_STATUS = TSS$REQUEST_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %REF(LIBRARY_ID), 7 %DESCR(REQUEST_NAME), 8 %REF(RECORD_1), 9 %REF(RECORD_2), a %REF(RECORD_n))

(continued on next page)

Table 5-6: TDMS Asynchronous Programming Calls in VAX FORTRAN (Cont.)

Call	Example
TSS$UNDECL_AFK_A	RET_STATUS = TSS$UNDECL_AFK_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %REF(KEY_ID))
TSS$WRITE_BRKTHRU_A	RET_STATUS = TSS$WRITE_BRKTHRU_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %DESCR(MESSAGE_TEXT), 7 %REF(BELL_FLAG))
TSS$WRITE_MSG_LINE_A	RET_STATUS = TSS$WRITE_MSG_LINE_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %DESCR(MESSAGE_TEXT))

Rules for Resolving Ambiguous Field References 6

The record field and form field names that you use in a request must:

- Exist in a record definition or form definition you refer to in the request
- Be unique within each record definition and form definition the request uses

In addition, in a %ALL mapping, at least one form field must have an identically named record field.

6.1 How to Make Field References Unique

Form field names are always unique because FDU does not allow the same name for two fields on a form. Because a request can use only one form in any particular request call, a form field name is always unique. If there is no active form or if the field you name is not in the active form, RDU displays an error message.

You must refer to record field names uniquely in a request. When you use a record field name in a request, RDU searches all of the records named in the RECORD IS instructions to find the field. Usually, a field name occurs only once in the set of records used by a request.

In some cases, however, a single record can contain fields with the same name. Or your request may use different records that have fields with the same name. For example, in Figure 6-1, the fields LAST, FIRST, and MID_INIT occur twice.

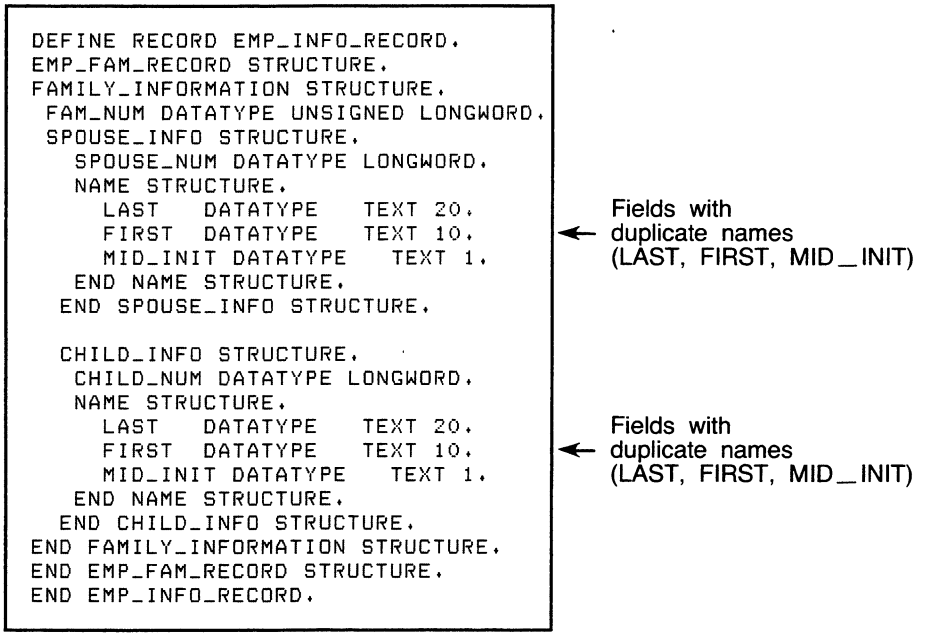


Figure 6-1: Referring to Record Fields with the Same Name

If a record contains fields with the same name, you must refer to them uniquely in your request. If you do not, RDU generates a message indicating that you have specified an ambiguous field reference. RDU does not create a mapping that contains ambiguous field references.

You can usually access fields that occur more than once in a single record, or more than once in a set of records used by a request, by qualifying the name with:

- One or more preceding group field names
- The record name

6.1.1 Using Group Field Names

You need specify only as many preceding group field names as necessary to make the reference unique. Each group field name must be followed by a period. For example, you can refer to the spouse's last name in EMP_INFO_RECORD as:

```
SPOUSE_INFO.LAST
```

You can also refer to the same field by using all the preceding group names:

```
EMP_FAM_RECORD.FAMILY_INFORMATION.SPOUSE_INFO.NAME.LAST
```

As long as the reference is unique, you can delete intervening group names. For instance, you can delete any intervening group names except SPOUSE_INFO:

```
SPOUSE_INFO.NAME.LAST
```

```
FAMILY_INFORMATION.SPOUSE_INFO.LAST
```

Note that using either the group field names EMP_FAM_RECORD or FAMILY_INFORMATION does not make the field reference to SPOUSE_INFO.LAST unique. For instance, the following references would be ambiguous because RDU cannot tell if you are referring to the child's name or the spouse's name:

```
EMP_FAM_RECORD.FAMILY_INFORMATION.LAST
```

```
FAMILY_INFORMATION.LAST
```

6.1.2 Using the Record Name

If you use a record name, it must be the first name in a field reference and must be followed by a period. RDU treats the record name as the top-level group field name.

Whether you specify the record name or not, RDU still searches all the records used by a request for a field reference.

You cannot use a record name to make a field reference within a single record unique. A record name, when used, qualifies all the fields under it in that record. For instance, in Figure 6-1, EMP_INFO_RECORD.LAST refers to both the child's and spouse's last names.

Using the record name to uniquely qualify fields can help, however, when you have two fields in separate records that have the same name. For example, the field LAST in Figure 6-2 is in two separate records used by a request.

```
DEFINE RECORD FAMILY_RECORD.
FAMILIES_RECORD STRUCTURE.
    NAME STRUCTURE.
      LAST    DATATYPE    TEXT 20.
        .
        .
        .

END  FAMILY_RECORD.
```

```
DEFINE RECORD DEPENDENT_RECORD.
FAMILIES_RECORD STRUCTURE.
     NAME STRUCTURE.
       LAST    DATATYPE    TEXT 20.
         .
         .
         .

END  DEPENDENT_RECORD.
```

Figure 6-2: Using Record Names to Make Field References Unique

You can refer to the two different fields named LAST by using the record names FAMILY_RECORD and DEPENDENT_RECORD:

```
DEPENDENT_RECORD.LAST

FAMILY_RECORD.LAST
```

If the record name is used, it must be the first qualifying name. The record name you use is either:

- The given name assigned in the RECORD IS instruction
- The unique name assigned by the WITH NAME modifier in the RECORD IS instruction

If you use the WITH NAME modifier in the RECORD IS instruction and you use a record name to refer to fields, you *must use* the unique name in field references. For instance, consider a request containing the following RECORD IS instruction:

```
RECORD IS CDD$TOP.OTHER_DIRECTORY.FAMILY_RECORD WITH NAME FAM;

RECORD IS FAMILY_RECORD;
```

To refer to a field named LAST that resides in each record:

- Use the unique name FAM to refer to the field in the record OTHER_DIRECTORY.FAMILY_RECORD
- Use the given record name FAMILY_RECORD to refer to the field in the record FAMILY_RECORD that resides in your default CDD directory

For example:

```
INPUT LAST TO (FAMILY_RECORD.LAST, FAM.LAST);
```

When you qualify a field name with a record name, as with any group field names, RDU allows you to delete intervening group name fields that are not necessary to make the reference unique. For example, you can refer to LAST as either:

```
FAMILY_RECORD.NAME.LAST
```

```
FAMILY_RECORD.LAST
```

6.1.3 Changing the Record Definition to Make References Unique

Because RDU does not require that you specify intervening fields, there are some fields RDU cannot access even when you use qualifying group field or record names.

For example, in the following record, there is no way to uniquely identify the first set of fields LAST, FIRST, and MID_INIT. Neither the record name EMP_INFO_RECORD nor the group name EMP_FAM_RECORD uniquely qualifies the first occurrence of the names LAST, FIRST, or MID_INIT.

```
DEFINE RECORD EMP_INFO_RECORD.
EMP_FAM_RECORD STRUCTURE.
   NAME STRUCTURE.
      LAST    DATATYPE    TEXT 20.
      FIRST   DATATYPE    TEXT 10.
      MID_INIT DATATYPE TEXT  1.
   END NAME STRUCTURE.
   CHILD_INFO STRUCTURE.
    NAME STRUCTURE.
      LAST    DATATYPE    TEXT 20.
      FIRST   DATATYPE    TEXT 10.
      MID_INIT DATATYPE TEXT  1.
    END NAME STRUCTURE.
   END CHILD_INFO STRUCTURE.
END EMP_FAM_RECORD STRUCTURE.
END EMP_INFO_RECORD.
```

First occurrence of fields with duplicate names (LAST, FIRST, MID_INIT)

Fields with duplicate names (LAST, FIRST, MID_INIT)

The following field specifications refer to both occurrences of the name LAST:

```
EMP_INFO_RECORD.LAST

EMP_FAM_RECORD.LAST
```

That is, both names refer to the first and second occurrences of the record field name LAST:

```
EMP_FAM_RECORD.NAME.LAST

EMP_FAM_RECORD.CHILD_INFO.NAME.LAST
```

To refer to the first occurrence of LAST, you can either:

- Change the name of one of the preceding group fields or the field itself to be unique:

```
DEFINE RECORD EMP_INFO_RECORD.
EMP_FAM_RECORD STRUCTURE.
  EMP_NAME STRUCTURE.                    <- Change group
    LAST    DATATYPE    TEXT 20.            name from NAME
    FIRST   DATATYPE    TEXT 10.            to EMP_NAME
    MID_INIT DATATYPE    TEXT 1.
  END EMP_NAME STRUCTURE.
  CHILD_INFO STRUCTURE.
  NAME STRUCTURE.
    LAST    DATATYPE    TEXT 20.
      .
      .
      .

END EMP_INFO_RECORD.
```

Then refer to the employee name using the new group structure name:

```
EMP_NAME.LAST
```

- Insert a new group name field in the record to allow you to make the reference unique:

```
DEFINE RECORD EMP_INFO_RECORD.
EMP_FAM_RECORD STRUCTURE.
EMPLOYEE_INFO STRUCTURE.                  <- Inserted
  NAME STRUCTURE.                            group field
    LAST    DATATYPE    TEXT 20.             EMPLOYEE_INFO
    FIRST   DATATYPE    TEXT 10.          <- Duplicate
    MID_INIT DATATYPE    TEXT 1.             names(LAST,
  END NAME STRUCTURE.                        FIRST, MID_INIT)
END EMPLOYEE_INFO STRUCTURE.
  CHILD_INFO STRUCTURE.
  NAME STRUCTURE.
    LAST    DATATYPE    TEXT 20.
      .
      .
      .

END EMP_INFO_RECORD.
```

Then refer to the employee name using the new group field name:

```
EMPLOYEE_INFO.LAST
```

Since RDU treats a record name as a preceding group field name, it is also possible that a field reference can be ambiguous because a record name exists also as a field name in the set of records used by a request.

For instance, in the following two records, the name FAMILY_RECORD is used first as a record name and second as a group field name. The following field reference is ambiguous:

```
FAMILY_RECORD.LAST
```

Because RDU searches all records even if you specify a record name, it finds the reference FAMILY_RECORD.LAST in both records.

```
DEFINE RECORD FAMILY_RECORD.
FAMILY_RECORD STRUCTURE.
NAME STRUCTURE.
   LAST    DATATYPE    TEXT 20.
      .
      .
      .

END  FAMILY_RECORD.
```

(continued on next page)

```
DEFINE RECORD DEPENDENT_RECORD.
FAMILY_RECORD STRUCTURE.
NAME STRUCTURE.
   LAST    DATATYPE    TEXT 20.
         .
         .
         .

END  DEPENDENT_RECORD.
```

In this case, even though you preface a field name with a record name, RDU is unable to resolve the reference. You must do one of the following:

1. Change the name of the first group field in the record definition
2. Change the name of the record in the record definition
3. Assign one of the record definitions a new name using the WITH NAME modifier in the RECORD IS instruction in the request that uses the record

Instruction Execution Order 7

The order in which instructions are specified in the request has no effect on the order in which TDMS executes the instructions.

First, TSS$REQUEST evaluates all CONTROL FIELD IS instructions. It attempts to match all control field 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.

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 keys can be pressed during any input operations.

- WAIT instruction.

Finally, TDMS evaluates and executes all RETURN operations.

VAX TDMS Input and Output Mapping Tables 8

8.1 Determining Data Types

The tables in this chapter show the validity of all possible combinations of input and output mappings for VAX TDMS applications. The input table shows mapping for data sent from a record field to a form field; the output table shows mapping for data sent from a record field to a form field.

Record field data types are determined by explicit statements of data types in the record definition. Form field data types are determined by:

- The picture characters of the form field
- Size field validators (for example, BYTE or UNSIGNED LONGWORD), if any, assigned to UNSIGNED NUMERIC fields

8.2 Determining Field Lengths

The length of a form field is determined by the number of picture characters in the field; field constants do not affect the length of a form field. For all record fields except those with a PACKED DECIMAL data type, the length of the record field is the number of bytes allocated for the field. For PACKED DECIMAL record fields, the length of the field is the number of digits in the field.

Any scale factors that are assigned to numeric form and record fields affect the length of a field for the purpose of determining the validity of mapping. *Add* the value of the scale factor to the length of a form or record field when using the input and output mapping tables. For example, if a 5-digit UNSIGNED NUMERIC form field (picture of 99999, regardless of any field constants) has a scale factor of 4, the field is considered to have a length of 9 characters. If a 5-digit SIGNED NUMERIC form field (picture of NNNNN) has a scale factor of -3, the field is considered to have a length of 2 characters.

8.3 How to Use These Tables

To use these tables, determine whether you want to verify the validity of an input mapping or an output mapping. Make sure that you know the data types and lengths of both the form and record fields between which you want to map data.

The validity code for each potential mapping is at the intersection of the form and record field data types on the tables. If the code is Y, the mapping is always permitted; if the code is N, the mapping is never permitted. The explanation of numeric codes is shown following each table.

Table 8-1: TDMS Input Mappings (Form Fields to Record Fields)

Y = Input mapping always permitted. N = Input mapping never permitted. 1-15 = Mapping permitted, subject to conditions. (See key following the table for explanation.)

Form Field Data Type	Record Field Data Types																				
	BU	WU	LU	B	W	L	Q	F	D	G	H	T	VT	GT	NU	NL	NLO	NR	NRO	NZ	P	ADT
UNSIGNED BYTE	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	2	2	1	2	2	2	2	2	2	2	N
UNSIGNED WORD	N	Y	Y	N	N	Y	Y	Y	Y	Y	Y	3	3	1	3	3	3	3	3	3	3	N
UNSIGNED LONGWORD	N	N	Y	N	N	N	Y	Y	Y	Y	Y	4	4	1	4	4	4	4	4	4	4	N
SIGNED BYTE	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	5	5	1	N	5	2	5	2	2	2	N
SIGNED WORD	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	6	6	1	N	6	3	6	3	3	3	N
SIGNED LONGWORD	N	N	N	N	N	Y	Y	Y	Y	Y	Y	7	7	1	N	7	4	7	4	4	4	N
SIGNED QUADWORD	N	N	N	N	N	N	Y	Y	Y	Y	Y	9	9	1	N	9	8	9	8	8	8	N
TEXT	N	N	N	N	N	N	N	N	N	N	N	1	1	1	N	N	N	N	N	N	N	N
UNSIGNED NUMERIC	10	11	12	10	11	12	13	14	14	Y	Y	1	1	15	1	1	1	1	1	1	1	N
SIGNED NUMERIC	N	N	N	10	11	12	13	14	14	Y	Y	1	1	15	N	1	1	1	1	1	1	N
DATE (Format 1)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	N	N	N	N	N	N	N	Y
DATE (Format 2)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	N	N	N	N	N	N	N	Y
DATE (Format 3)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	1	N	N	N	N	N	1	Y
DATE (Format 4)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	1	N	N	N	N	N	1	Y
DATE (Format 5)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	N	N	N	N	N	N	N	Y
TIME (Format 1)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	1	N	N	N	N	N	1	Y
TIME (Format 2)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	N	N	N	N	N	N	N	Y

Record Field Data Types:

BU – UNSIGNED BYTE
WU – UNSIGNED WORD
LU – UNSIGNED LONGWORD
B – SIGNED BYTE
W – SIGNED WORD
L – SIGNED LONGWORD
Q – SIGNED QUADWORD
F – F_FLOATING
D – D_FLOATING
G – G_FLOATING
H – H_FLOATING
T – TEXT
VT – VARYING TEXT
GT – GROUP TEXT (non-elementary item)
NU – UNSIGNED NUMERIC
NL – LEFT SEPARATE NUMERIC
NLO – LEFT OVERPUNCHED NUMERIC
NR – RIGHT SEPARATE NUMERIC
NRO – RIGHT OVERPUNCHED NUMERIC
NZ – SIGNED NUMERIC (zoned sign)
P – PACKED DECIMAL
ADT – DATE

Key for Input Mappings:

1 – Permitted only if length of record field is greater than or equal to length of form field.
2 – Permitted only if record field has length of at least 3 bytes.
3 – Permitted only if record field has length of at least 5 bytes.
4 – Permitted only if record field has length of at least 10 bytes.
5 – Permitted only if record field has length of at least 4 bytes.
6 – Permitted only if record field has length of at least 6 bytes.
7 – Permitted only if record field has length of at least 11 bytes.
8 – Permitted only if record field has length of at least 19 bytes.
9 – Permitted only if record field has length of at least 20 bytes.
10 – Permitted only if form field has no more than 2 characters.
11 – Permitted only if form field has no more than 4 characters.
12 – Permitted only if form field has no more than 9 characters.
13 – Permitted only if form field has no more than 18 characters.
14 – Permitted only if form field has no more than 38 characters.
15 – Permitted only if length of record field is greater than or equal to length of form field; RDU also issues a warning message possible conversion error on group move.

Table 8-2: TDMS Output Mappings (Record Fields to Form Fields)

Y = Output mapping always permitted. N = Output mapping never permitted. 1-29 = Mapping permitted, subject to conditions. (See key following the table for explanation.)

Form Field Data Type	Record Field Data Types																					
	BU	**WU**	**LU**	**B**	**W**	**L**	**Q**	**F**	**D**	**G**	**H**	**T**	**VT**	**GT**	**NU**	**NL**	**NLO**	**NR**	**NRO**	**NZ**	**P**	**ADT**
UNSIGNED BYTE	Y	15	15	16	1	1	1	N	N	N	N	N	N	14	18	25	25	25	25	25	25	N
UNSIGNED WORD	Y	Y	15	16	16	1	1	N	N	N	N	N	N	14	19	26	26	26	26	26	26	N
UNSIGNED LONGWORD	Y	Y	Y	16	16	16	1	N	N	N	N	N	N	14	20	27	27	27	27	27	27	N
SIGNED BYTE	15	15	15	Y	15	15	15	N	N	N	N	N	N	14	18	18	18	18	18	18	18	N
SIGNED WORD	Y	15	15	Y	Y	15	15	N	N	N	N	N	N	14	19	19	19	19	19	19	19	N
SIGNED LONGWORD	Y	Y	15	Y	Y	Y	15	N	N	N	N	N	N	14	20	20	20	20	20	20	20	N
SIGNED QUADWORD	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	14	21	21	21	21	21	21	21	N
TEXT	2	3	4	5	6	7	12	17	17	17	17	13	13	13	13	13	24	13	24	24	24	N
UNSIGNED NUMERIC	2	3	4	8	9	10	11	N	N	N	N	29	29	14	13	23	28	23	28	28	28	N
SIGNED NUMERIC	2	3	4	5	6	7	12	N	N	N	N	29	29	14	13	13	24	13	24	24	24	N
DATE (Format 1)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	N	N	N	N	N	N	N	Y
DATE (Format 2)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	N	N	N	N	N	N	N	Y
DATE (Format 3)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	13	N	N	N	N	N	13	Y
DATE (Format 4)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	13	N	N	N	N	N	13	Y
DATE (Format 5)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	N	N	N	N	N	N	N	Y
TIME (Format 1)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	13	N	N	N	N	N	13	Y
TIME (Format 2)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	N	N	N	N	N	N	N	Y

Record Field Data Types:

BU – UNSIGNED BYTE
WU – UNSIGNED WORD
LU – UNSIGNED LONGWORD
B – SIGNED BYTE
W – SIGNED WORD
L – SIGNED LONGWORD
Q – SIGNED QUADWORD
F – F_FLOATING
D – D_FLOATING
G – G_FLOATING
H – H_FLOATING
T – TEXT
VT – VARYING TEXT
GT – GROUP TEXT (non-elementary item)
NU – UNSIGNED NUMERIC
NL – LEFT SEPARATE NUMERIC
NLO – LEFT OVERPUNCHED NUMERIC
NR – RIGHT SEPARATE NUMERIC
NRO – RIGHT OVERPUNCHED NUMERIC
NZ – SIGNED NUMERIC (zoned sign)
P – PACKED DECIMAL
ADT – DATE

Key for Output Mappings:

1 – Possible overflow; negative value in record field produces run-time error. RDU issues warning message.
2 – RDU issues a warning message if form field is less than 3 characters (possible overflow).
3 – RDU issues a warning message if form field is less than 5 characters (possible overflow).
4 – RDU issues a warning message if form field is less than 10 characters (possible overflow).
5 – RDU issues a warning message if form field is less than 4 characters (possible overflow).
6 – RDU issues a warning message if form field is less than 6 characters (possible overflow).
7 – RDU issues a warning message if form field is less than 11 characters (possible overflow).
8 – Possible overflow if form field is less than 3 characters; negative value in record produces run-time error. RDU issues a warning message.
9 – Possible overflow if form field is less than 5 characters; negative value in record produces run-time error. RDU issues a warning message.
10 – Possible overflow if form field is less than 10 characters; negative value in record produces run-time error. RDU issues a warning message.
11 – Possible overflow if form field is less than 19 characters; negative value in record produces run-time error. RDU issues a warning message.
12 – RDU issues a warning message if form field is less than 20 characters (possible overflow).
13 – RDU issues a warning message if length of record field is greater than length of form field (possible overflow).
14 – RDU issues a warning message if length of record field is greater than length of form field (possible overflow, possible error on group move).
15 – RDU issues a warning message (possible overflow).
16 – RDU issues a warning message; negative value in record field produces run-time error.
17 – Permitted only if form field is at least 8 characters in length; display format is in scientific notation.
18 – RDU issues a warning message if record field is more than 2 bytes.
19 – RDU issues a warning message if record field is more than 4 bytes.
20 – RDU issues a warning message if record field is more than 9 bytes.
21 – RDU issues a warning message if record field is more than 18 bytes.
22 – RDU issues a warning message if record field is more than 38 bytes.
23 – RDU issues a warning message if record field is more than 1 character longer than form field. Negative value in record produces run-time error.
24 – RDU issues a warning message if form field is not at least 1 character longer than record field.
25 – RDU issues a warning message if record field is more than 2 bytes. Negative value in record field produces run-time error.
26 – RDU issues a warning message if record field is more than 4 bytes. Negative value in record field produces run-time error.
27 – RDU issues a warning message if record field is more than 10 bytes. Negative value in record field produces run-time error.
28 – RDU issues a warning message if length of record field is greater than length of form field (possible overflow). Negative value in record field produces run-time error.
29 – RDU issues warning message; data output to form may be inconsistent with form field picture string.

FDU and Field Validator Error Messages A

This appendix lists error messages that FDU generates for FDU commands and the field validator portion of the form editor.

A.1 FDU-Level Error Messages

This section lists the error messages alphabetically by the mnemonic. The %FDU is not shown and the severity level code is called out separately for your convenience. For example, the error message %FDU-E-BAD_TERMINAL is listed under BAD_TERMINAL, the first message in the list. The text that immediately follows the error message mnemonic (for example, "unable to display form on this terminal") represents the actual text that appears in the message at run time.

The error message format is:

%FDU-N-mnemonic, message

FDU — Is the facility name.

N — Is a letter indicating the severity of the error, as follows:

- S — Successful completion.
- I — Information only. The request is created and stored in the CDD.
- W — Warning. The request is created and stored in the CDD, but it might contain instructions that cause run-time errors.
- E — Error. The request is not created and stored in the CDD.

F Fatal severe error. The request is not created and stored in the CDD.

mnemonic Is a three- to nine- character string that identifies the error.

message Identifies the error.

Note

Most error messages are prefixed by %FDU. However, some %FDU messages are followed by messages with a %CMU prefix. These messages provide additional information about the cause of the error. Messages with a prefix other than %FDU or %CMU reflect errors detected by some standard VMS facility which the application program uses. For example, a message prefixed by the %RMS facility code was generated by the RMS facility. For information about these messages, please see the *VMS System Messages and Recovery Procedures Manual.*

BAD_TERMINAL **unable to display form on this terminal**

Severity: Error (E)

Explanation: FDU is unable to display a form from the CDD onto the terminal.

User Action: Make sure that the terminal is a type supported by VAX TDMS.

CDD_FORM_EMPTY **CDD Form contains no valid information**

Severity: Error (E)

Explanation: The form definition contains no information.

User Action: Recreate the form definition with background text and/or at least one field.

FMS_FIELD_UAR	**Field UAR information has been ignored for this form**
Severity:	Information (I)
Explanation:	The FMS version 2 form file contains User Action Routine information that TDMS does not support. This information has been ignored.
User Action:	None (informational message).

FMS_HELPUAR	**Form HELP UAR information has been ignored for this form**
Severity:	Information (I)
Explanation:	The FMS version 2 form file contains User Action Routine information that TDMS does not support. This information has been ignored.
User Action:	None (informational message).

FMS_KEYUAR	**Key UAR information has been ignored for this form**
Severity:	Information (I)
Explanation:	The FMS version 2 form file contains User Action Routine information that TDMS does not support. This information has been ignored.
User Action:	None (informational message).

FMS_NAMEDATA	**NAMED DATA information has been ignored for this form**
Severity:	Information (I)
Explanation:	The FMS version 2 form file contains named data information that TDMS does not support. This information has been ignored.
User Action:	None (informational message).

FMS_SUP_ONLY	**SUPERVISOR-ONLY information has been ignored for this form**
Severity:	Information (I)
Explanation:	The FMS version 2 form file contains Supervisor Only information that TDMS does not support. This information has been ignored.
User Action:	None (informational message).
FRMCOPIED	**Form <formname> copied to <new formname>**
Severity:	Success (S)
Explanation:	Logging was enabled and the form was successfully copied.
User Action:	None (success message).
FRMCREATE	**Form <formname> created**
Severity:	Success (S)
Explanation:	Logging was enabled and the form was successfully created.
User Action:	None (success message).
FRMDELETE	**Form <formname> deleted**
Severity:	Success (S)
Explanation:	Logging was enabled and the form was successfully deleted.
User Action:	None (success message).
FRMMODIFY	**Form <formname> modified**
Severity:	Success (S)
Explanation:	Logging was enabled and the form was successfully modified.
User Action:	None (success message).

FRMREPLAC	**Form <formname> replaced**
Severity:	Success (S)
Explanation:	Logging was enabled and the form was successfully replaced.
User Action:	None (success message).
ILLWINDOW	**Illegal window attribute value**
Severity:	Error (E)
Explanation:	Attempted to list a form with an invalid form size (for example, first line, last line or column width).
User Action:	Recreate the form definition.
NAMEDATA	**Named data information has been ignored for this form**
Severity:	Information (I)
Explanation:	The FMS version 1 form file contains named data information that TDMS does not support. This information has been ignored.
User Action:	None (informational message).
NO_BFDHEAD	**Format of Input .FRM file invalid**
Severity:	Error (E)
Explanation:	The FMS V2 form file is invalid.
User Action:	Make sure that the file that you specified is a valid VAX FMS V2 form file.

NO_CDD_REQD	**Form not entered into CDD - not required by user**
Severity:	Error (E)
Explanation:	The form is not being stored in the CDD.
User Action:	None.
NO_FDV_WORKSPACE	**Unable to set up Forms Driver work-area**
Severity:	Error (E)
Explanation:	An error occurred trying to allocate memory for the form driver internal work area.
User Action:	Ask your system manager to increase your page fault quota or the virtual page count system parameter.
NO_FEDIO	**Unable to activate Form Editor I/O**
Severity:	Error (E)
Explanation:	Either the terminal is not a VT100-compatible terminal or the VMS assign channel system service failed.
User Action:	Make sure that the terminal that you are using is a supported terminal type.
NO_FED_WORKSPACE	**Unable to set up Form Editor work-area**
Severity:	Error (E)
Explanation:	An error occurred trying to allocate memory for the form editor internal work area.
User Action:	Ask your system manager to increase your page fault quota or the virtual page count system parameter.
NO_FIELD_NAME	**An unnamed field has been detected in this form**
Severity:	Severe (F)
Explanation:	A field on the form has not been assigned a field name.
User Action:	Recreate the form definition, and make sure to give a unique name to each field.

NOFORM	**Form file must be specified on CREATE or REPLACE a form**
Severity:	Error (E)
Explanation:	A form file name was not specified with the /v1 qualifier.
User Action:	Reenter the command and include the name of a valid form file with the /FORM_FILE= qualifier.
NO_FORM_IMAGE	**No Screen Image has been created for this form**
Severity:	Error (E)
Explanation:	Attempted to store a form definition with neither background text nor fields.
User Action:	Recreate the form. You must use the Layout phase to include at least one character that is background text, a field, or a video attribute.
NOFRMCOP	**Form < formname > not copied**
Severity:	Error (E)
Explanation:	An error occurred while copying the form. The form has not been copied.
User Action:	This message follows one or more other FDU error messages. Check the other error messages to determine the recommended user action.
NOFRMCRE	**Form < formname > not created**
Severity:	Error (E)
Explanation:	An error occurred while creating the form. The form has not been created.
User Action:	This message follows one or more other FDU error messages. Check the other error messages to determine the recommended user action.

NOFRMDEL	**Form <formname> not deleted**
Severity:	Error (E)
Explanation:	An error occurred while deleting the form. The form has not been deleted.
User Action:	This message follows one or more other FDU error messages. Check the other error messages to determine the recommended user action.
NOFRMMOD	**Form <formname> not modified**
Severity:	Error (E)
Explanation:	An error occurred while modifying the form. The form has not been modified.
User Action:	This message follows one or more other FDU error messages. Check the other error messages to determine the recommended user action.
NOFRMREP	**Form <formname> not replaced**
Severity:	Error (E)
Explanation:	An error occurred while replacing the form. The form has not been replaced.
User Action:	This message follows one or more other FDU error messages. Check the other error messages to determine the recommended user action.
NO_INBFD	**Unable to set up Form Editor data structures**
Severity:	Error (E)
Explanation:	An error occurred while attempting to allocate memory for internal data structures.
User Action:	Ask your system manager to increase your page fault quota or the virtual page count system parameter.

NO_OUTBFD **Unable to get memory for Output .FRM file build**

Severity: Error (E)

Explanation: An error occurred trying to allocate memory for internal structures for the form that was being created, modified, or replaced.

User Action: Ask your system manager to increase your page fault quota or the virtual page count system parameter.

REQATTNFND **Required attribute < attribute name > not found**

Severity: Error (E)

Explanation: An internal error occurred attempting to create the form to be stored in the CDD.

User Action: Send in an SPR with the problem description and a DMU backup of the form that caused this error. Recreate the form definition.

SUPERMOD **Supervisor modification has been ignored for this form**

Severity: Information (I)

Explanation: The FMS version 1 form file contains supervisor only information that TDMS does not support. This information has been ignored.

User Action: None (informational message).

SUPERPROT **Supervisor protection has been ignored for field**

Severity: Information (I)

Explanation: The FMS version 1 form file contains supervisor only information that TDMS does not support. This information has been ignored.

User Action: None (informational message).

A.2 Field Validator Error Messages

The following are error messages that can occur when assigning a field validator to a field:

"Cannot have abbrev. char. equal to fill char"

Explanation: Attempted to assign the abbreviation character for the choice validator to be the same as the fill character.

"Cannot have abbreviation length AND character"

Explanation: Attempted to assign an abbreviation length and abbreviation character for the choice validator.

"field already has an fvp"

Explanation: Attempted to add a field validator to a field that already has a field validator.

"field is not numeric"

Explanation: Attempted to add a size validator or check digit validator on a field that is not numeric (picture of all 9's or all N's).

"field is not unsigned numeric"

Explanation: Attempted to assign a check digit validator on a field that is not unsigned numeric (picture of all 9's).

"field is too small for fvp"

Explanation: Attempted to assign a size validator to a field that is not large enough for the validator (for example, attempting to assign the UNSIGNED BYTE validator to a numeric field picture with fewer than 3 digits.)

"high range is less than low range"

Explanation: Attempted to specify a Range validator with a high range that is less than the low range.

"illegal input--enter 1-14"

Explanation: Attempted to enter an invalid selection number in the field validator menu form.

"internal range conversion"

Explanation: An internal error occurred while attempting to determine if the low range is less than the high range. The low range must be less than or equal to the high range.

"Low range must be specified"

Explanation: A low range must always be assigned on a field with a range field validator.

"Too many choice values"

Explanation: Attempted to assign more than 255 choices to a field.

"Too many range values"

Explanation: Attempted to assign more than 127 range values to a field.

RDU Error Messages B

RDU checks request text for syntax errors and displays error messages when it encounters an error. It usually continues to take additional request instructions until you enter the final instruction END DEFINITION followed by a semicolon. RDU then makes a second check of the text for mapping errors.

Under some circumstances, RDU may encounter syntax errors in your request text from which it cannot recover. In this case, it returns to the RDU> prompt and does not check for mapping errors.

The error message format is:

%RDU-N-mnemonic, message

RDU — Is the facility name.

N — Is a letter indicating the severity of the error, as follows:

- S — Successful completion.
- I — Information only. The request is created and stored in the CDD.
- W — Warning. The request is created and stored in the CDD, but it might contain instructions that cause run-time errors.
- E — Error. The request is not created and stored in the CDD.
- F — Fatal severe error. The request is not created and stored in the CDD.

mnemonic Is a three- to nine- character string that identifies the error.

message Identifies the error.

This appendix lists the error messages alphabetically by the mnemonic. The %RDU is not shown and the severity level code is called out separately for your convenience. For example, the error message %RDU-E-ACTDEPRNG is listed under ACTDEPRNG, the first message in the list. The explanation indicates whether the error is a syntax message or whether it is mapping message. Text enclosed in angle brackets (for example, <text-of-message> or <number>) represents the actual text or number that appears in the message at run time.

Note

Most error messages are prefixed by %RDU. A few continuation messages will appear with the prefix %CMU. These messages provide additional information about the cause of the error. Messages prefixed with a facility name other than %RDU and %CMU reflect errors detected by some standard VMS facility that the program uses. For example, a message prefixed by the %RMS facility code is generated by the RMS facility. For information about these messages, see the *VAX/VMS System Messages and Recovery Procedures Manual.*

ACTDEPRNG **too many active dependent ranges**

Severity: Error (E)

Explanation: A syntax message. More than two active dependent ranges are specified within a control field or nested control fields.

User Action: Reduce the number of dependent ranges within the control field or nested control fields to two.

ACTFRMILL **form is not in the FORM IS list**

Severity: Error (E)

Explanation: A syntax and mapping message. The form used in a USE FORM or DISPLAY FORM instruction has not been declared in a FORM IS instruction in the header part of the request.

User Action: Declare the form using a FORM IS instruction in the header part of the request.

AMBFLDSPC **ambiguous field specification**

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

Explanation: A mapping message. A record field named in an OUTPUT, INPUT, or RETURN mapping instruction matches more than one record field within the records used by the request.

User Action: Make the record field name unique by using as many preceding group names as are necessary or by using the record name.

BITFLDUNS **field is a bit-field - not supported**

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

Explanation: A mapping message. The field is defined as a bit-field, which is not valid in TDMS.

User Action: Redefine the field to be one of the data types valid in TDMS.

CHNGBND **changing the subscript bounds of the array from <n:m> to <n:m>**

Severity: Explicit: Warning (W)

Explanation: A syntax message. The lower bound of a record array has some value other than 1. RDU has adjusted both the lower

and upper bounds of this dimension so that the lower bound is 1. The array contains the same number of elements.

User Action: None necessary. In the request reference to the array, you always reference the first element using a subscript value of 1.

CNDNEGSRC **mapping may produce conversion error if source is negative**

Severity: Explicit: Error (E) Input or return mappings
Warning (W) Output mappings

%ALL: Information (I)

Explanation: A mapping message. The mapping may produce a data conversion error if the source is negative.

User Action: Make the signs of the sending and receiving fields compatible.

CONDCONVN **mapping may produce a data conversion error on group move**

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

Explanation: A mapping message. A data conversion error may result at run time if you are mapping a group field.

User Action: Check your mapping instructions that map group fields to make sure they are correct.

CONDOVFLW **mapping may produce overflow condition during data conversion**

Severity: Explicit: Error (E) Input or return mappings
Warning (W) Output mappings

%ALL: Information (I)

Explanation: A mapping message. The mapping may produce an overflow condition during data conversion.

User Action: Make the sending and receiving field data types, lengths (including scale factor), and sizes compatible.

CTLNOTTEX **control field is not a text field**

Severity: Error (E)

Explanation: A mapping message. The control field is not a TEXT or VARYING TEXT data type.

User Action: Make the control field data type text.

DECOVF **packed decimal overflow error**

Severity: Explicit: Error (E) Input or return mappings
Warning (W) Output mappings

%ALL: Information (I)

Explanation: A mapping message. A data type conversion error occurs when mapping a literal string to a form or record field because of a packed decimal overflow.

User Action: Make the sending and receiving field lengths and sizes compatible.

DEPNAMACT **dependent name is already active**

Severity: Error (E)

Explanation: A syntax message. You made an attempt to explicitly assign a dependent name to a dependent range (by using a %LINE= or %ENTRY= in the CONTROL FIELD IS phrase). The attempt fails because the dependent range of the control field array is already assigned.

User Action: Assign to the dependent range a dependent name that is not active, or assign the dependent name in another control field.

DEPREFINV **dependent name is referenced outside a conditional request**

Severity: Error (E)

Explanation: A syntax message. You attempted to use a dependent name, %LINE or %ENTRY, in a mapping instruction that

is outside a conditional instruction (that uses an array with a dependent range as a control field).

User Action: Remove the dependent name from the field reference or move the mapping instruction inside of a conditional instruction that uses an array with a dependent range.

DSTLENGEQ **Destination length must be greater than or equal to <number>**

Severity: Explicit: Error (E) Input or return mappings
Warning (W) Output mappings

%ALL: Information (I)

Explanation: A mapping message. A data type conversion error occurred because the length of the receiving field (including the scale factor) is insufficient to hold the largest sending value. The message states how long the receiving field must be.

User Action: Change the length of the receiving field (including the scale factor) to be equal to or greater than the length specified in the message.

DSTMORDIM **destination field has more dimensions than source**

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

Explanation: A mapping and syntax message. The receiving field of an OUTPUT, INPUT, or RETURN mapping instruction has more dimensions than the sending field.

User Action: Make the number of dimensions in the receiving field of the mapping instruction the same number as in the sending field.

DSTMORENT **destination field has more entries than source**

Severity: Error (E)

Explanation: A mapping and syntax message. The receiving field reference contains a greater number of elements than the sending field reference.

User Action: Adjust the receiving field reference to contain the same number of elements as the sending field reference.

ERRCLSINCL **error closing include file**

Severity: Error (E)

Explanation: A syntax message. An error occurred when TDMS attempted to close a %INCLUDE file or to continue checking the request text from a %INCLUDE file. The message is output with other messages explaining the problem.

User Action: Correct the error specified in the accompanying message.

ERRDPAR **error during instruction processing**

Severity: Error (E)

Explanation: A syntax message. This message appears with other syntax messages explaining the specific error.

User Action: Correct the syntax error specified in the accompanying message.

ERROPNINCL **error opening include file**

Severity: Error (E)

Explanation: A syntax message. A VAX RMS error occurred when RDU attempted to open the %INCLUDE file. This message appears with other messages explaining the specific error.

User Action: Correct the error explained in the accompanying error message.

ERRRLBFIL **error on library file**

Severity: Error (E)

Explanation: A mapping message. A VAX RMS error occurred when RDU attempted to write to or read from the request library file. This message is followed immediately by a VAX RMS message.

User Action: See the VAX RMS documentation for a description of the error and user action.

FLTOVF **floating overflow error**

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

Explanation: A mapping message. A data type conversion error occurred due to floating overflow when you mapped a literal string to a form or record field.

User Action: Make the sizes and lengths of sending and receiving fields compatible. See the documentation on LIB$CVT_DX_DX in the *VAX/VMS Run-Time Library Routines Reference Manual.*

FLTUND **floating underflow error**

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

Explanation: A mapping message. A data type conversion error occurred due to floating underflow when mapping a literal string to a form or record field. See the documentation on LIB$CVT_DX_DX in the *VAX/VMS Run-Time Library Routines Reference Manual.*

User Action: Make the lengths and sizes of sending and receiving fields compatible.

FNDSYNTER **incorrect syntax <incorrect-instruction>**

Severity: Error (E)

Explanation: A syntax message. An instruction contains incorrect syntax.

User Action: Correct the syntax error.

FRMNOFLDS **form has no fields**

Severity: Error (E)

Explanation: A mapping message. Mappings reference a form that has no fields.

User Action: Check to make sure that the form name is correct, or redefine the form to contain fields.

FRMNOINP **form field is a DISPLAY-ONLY field - input mappings illegal**

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

Explanation: A mapping message. Input mappings reference a field that has been defined as Display Only in the form definition.

User Action: Remove the Display Only attribute from the field or remove the mapping instructions.

HLPFRMNOT **help form will not be available in the request library**

Severity: Warning (W)

Explanation: A mapping message. Some error occurred when RDU accessed a help form in the CDD for inclusion in the request library file. This help form will not be included in the request library file. This message is preceded by a message giving the precise problem.

User Action: Correct the problem identified by the accompanying message or remove the help form name from the form which calls the HELP forms.

ILLCHAR **illegal character in text**

Severity: Error (E)

Explanation: A syntax message. An illegal character or escape sequence (for example, line feed or form feed) appears somewhere in a request instruction line.

User Action: Remove the illegal character from the request instruction line.

ILLDEPRNG **illegal nested dependent range**

Severity: Error (E)

Explanation: A mapping message. A second array with a dependent range is used within a single CONTROL FIELD IS instruction.

User Action: End the first CONTROL FIELD IS instruction with an END CONTROL FIELD phrase before declaring a second CONTROL FIELD IS instruction which uses an array as a control field.

ILLDESCCHR **illegal character in description**

Severity: Error (E)

Explanation: A syntax message. An illegal character or escape sequence (for example, form feed or line feed) is in the text of a DESCRIPTION instruction.

User Action: Remove the illegal character from the descriptive text.

ILLDSTDAT **unsupported datatype in destination of mapping**

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

Explanation: A mapping message. The data type of the receiving field is not supported by TDMS, or the data types of the sending and receiving fields are incompatible.

User Action: Change the data type of the receiving field to one that TDMS supports, or make the data types of the sending and receiving fields compatible.

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 GOLD 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 GOLD instruction.

ILLKEYLEN **program key can be only 1 character long**

Severity: Error (E)

Explanation: A mapping message. You defined a program request key that is more than one character long.

User Action: Specify a program request key that is one character in length, for example, GOLD "T" instead of GOLD "TZ".

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 move than 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**

Severity: Error (E)

Explanation: A mapping message. You specified a program request key that is not one of the keys valid in a PROGRAM KEY IS instruction.

User Action: Specify a program request key that is one of the keys valid in a PROGRAM KEY IS 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 datatype in source of mapping**

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

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

User Action: Change the data type of the sending field to one that is supported by TDMS.

ILLSTRGCHR **illegal character in quoted string**

Severity: Error (E)

Explanation: A syntax message. An illegal character or escape sequence is in the quoted string (for example, a line feed or form feed).

User Action: Remove the illegal character from the quoted string.

ILLSUBVAL **Subscript values must be greater than zero**

Severity: Error (E)

Explanation: A syntax message. A mapping contains a subscript value that is less than or equal to zero. All subscripts in a request must be greater than zero.

User Action: Change the subscript to be greater than zero.

ILLUNQNAM **illegal character in given name or unique name < text >**

Severity: Error (E)

Explanation: A syntax message. The given name or unique name specified in the message is not a legal CDD name.

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

ILLWLDCRD **illegal wildcard character in file specification**

Severity: Error (E)

Explanation: A syntax message. A wildcard character appears in the file specification on a FILE IS instruction in a request library definition.

User Action: Remove the wildcard character from the file specification.

INPOPNPMT **Mapping may introduce invalid data during datatype conversion**

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

Explanation: A mapping message. In an INPUT or RETURN mapping, the data type of the record field is TEXT (T or VT) and the data type of the form field is NUMERIC MIX (NX) or UNSIGNED NUMERIC (NU). This mapping is not valid.

User Action: Change the data type of the form field and/or the record field in the INPUT or RETURN mapping.

INTOVF **integer overflow error**

Severity: Explicit: Error (E)

Explanation: A mapping message. A data type conversion error occurred due to integer overflow when you mapped a literal string to a form or record field. See the documentation on LIB$CVT_DX_DX in the *VAX/VMS Run-Time Library Routines Reference Manual.*

User Action: Make the sending and receiving field lengths and sizes compatible.

INVCONFLD **invalid CONTROL FIELD specification**

Severity: Error (E)

Explanation: A mapping message. This is a general message stating that there is something wrong with a control field specification. It appears with other control field error messages.

User Action: Check the control field specification to make sure it is correct.

INVCVT **negative source value and unsigned destination datatype in mapping**

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

Explanation: A mapping message. A data type conversion error occurred when mapping a literal string to a form or record field,

because the sending field value is negative and the receiving field data type is unsigned. See the documentation on LIB$CVT_DX_DX in the *VAX/VMS Run-Time Library Routines Reference Manual.*

User Action: Make the signs of the sending and receiving fields compatible.

INVFLDREF **invalid field reference**

Severity: Error (E)

Explanation: A mapping message. A field reference contains syntactic errors (for example, an incorrect number of dimensions or subscript limits). This message is always displayed with other field reference messages that provide more information about the specific error.

User Action: Check the field reference to make sure that it is legal.

INVFRMDEF **invalid form definition**

Severity: Error (E)

Explanation: A mapping message. The form definition in a FORM IS instruction has some characteristic that prevents RDU from processing it. This message appears with other messages that indicate the particular problem with the form.

User Action: Fix the error in the form indicated by the accompanying error message and replace the form in the CDD.

INVFRMOFF **invalid form OFFSET value - form takes lines <n to m>**

Severity: Error (E)

Explanation: A mapping message. The WITH OFFSET value given in a USE FORM or DISPLAY FORM instruction would put part of the form off the terminal screen.

User Action: Adjust the OFFSET value so that no part of the form is off the terminal screen.

INVNBDS **invalid Numeric Byte Data String (NBDS)**

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

Explanation: A mapping message. A data type conversion error occurred when you mapped a literal string to a form or record field, because there was an invalid character in the string or the value was outside the range that can be contained by the receiving field. See the documentation on LIB$CVT_DX_DX in the *VAX/VMS Run-Time Library Routines Reference Manual.*

User Action: Remove the invalid character from the string or make the sizes and lengths of the sending and receiving fields compatible.

INVRECDEF **invalid record definition**

Severity: Error (E)

Explanation: A mapping message. The record definition used in a RECORD IS instruction has some characteristic that prevents RDU from processing it. For example, if the record contains an array with a variable lower bound, RDU cannot process the record. This message appears with other messages (for example, LOWBNDMIS) that indicate the particular problem with the record.

User Action: Correct the problem in the record.

KEYMAYNOT **this key may not execute as expected**

Severity: Warning (W)

Explanation: A mapping message. The program request key may not execute as expected because the request contains no KEYPAD IS instruction. This message appears with the KPMNOTSET message.

User Action: Include a KEYPAD IS instruction in your request specifying that the keypad should be set to Application mode.

KPMNOTSET **keypad mode has not been set to Application mode in this request**

Severity: Warning (W)

Explanation: A mapping message. A PROGRAM KEY IS KEYPAD instruction was used in a request without a KEYPAD

(MODE) IS Application instruction. This message appears with the KEYMAYNOT message.

User Action: Include a KEYPAD MODE IS instruction in your request.

LOWBNDMIS **lower bound for dimension is missing - dimension has variable lower bound**

Severity: Error (E)

Explanation: A mapping message. The lower bound of a record field definition is missing.

User Action: RDU cannot process record definitions that have arrays with variable lower bounds. Add the lower bound to the field reference.

MISPELKWD **misspelled keyword < incorrect-keyword > ; should be < correct-keyword >**

Severity: Error (E)

Explanation: A syntax message. A keyword (a required word in an instruction) was misspelled.

User Action: Correct the spelling of the keyword.

MULPRKINP **more than one RETURN action for PROGRAM KEY**

Severity: Error (E)

Explanation: A mapping message. A PROGRAM KEY IS instruction contains more than one RETURN instruction.

User Action: Reduce the number of RETURN instructions in the PROGRAM KEY IS instruction to one.

MULPRKOUT **more than one OUTPUT or MESSAGE LINE action for PROGRAM KEY**

Severity: Error (E)

Explanation: A mapping message. A PROGRAM KEY IS instruction contains more than one OUTPUT or MESSAGE LINE IS instruction.

User Action: Specify only one OUTPUT or MESSAGE LINE IS instruction in the PROGRAM KEY IS instruction.

MULTFILIS **multiple FILE IS instructions found - ignoring instruction**

Severity: Warning (W)

Explanation: A syntax message. The request library definition contains more than one FILE IS instruction. Only the first FILE IS instruction is executed.

User Action: Remove the excess FILE IS instructions from your request library definition.

NAMNOTACT **dependent name is not active**

Severity: Error (E)

Explanation: A syntax message. A mapping instruction uses a dependent name (%LINE or %ENTRY) that has not been assigned to a dependent range.

User Action: Assign the dependent name to a dependent range by using a control field array with a dependent range or remove the dependent name from the field reference that is in error.

NOACTFRM **no active form declared with DISPLAY FORM or USE FORM**

Severity: Error (E)

Explanation: A mapping message. The request specifies a reference to a form field, but the request has not specified an active form in the current context.

User Action: Make sure the request declares an active form in the current context.

NOBELLNUM **cannot specify a number of bells with NO BELL syntax**

Severity: Error (E)

Explanation: A syntax message. A number of bells is specified with the NO RING BELL instruction.

User Action: Remove the number of bells from the NO RING BELL instruction.

NODEFRLB **no default .RLB file defined for library**

Severity: Error (E)

Explanation: A mapping message. You did not specify a request library file name when you issued the BUILD LIBRARY command and the request library definition does not contain a FILE IS instruction.

User Action: Specify the library file name in the FILE IS instruction in the request library definition or issue the BUILD LIBRARY command using the file-name parameter.

NOINCLFIL **no include file specified**

Severity: Error (E)

Explanation: A syntax message. The file name in a %INCLUDE instruction is not enclosed in quotation marks or the file name is missing.

User Action: Enclose the file name in the %INCLUDE instruction in quotation marks or supply the file name.

NOKEYPUNC **missing <keyword or punctuation> before <keyword or punctuation>**

Severity: Error (E)

Explanation: A syntax message. A keyword or punctuation is missing in an instruction.

User Action: Include the missing keyword or punctuation in the instruction.

NOMAPCRE **no mappings created**

Severity: Error (E)

Explanation: A mapping message. An error occurred when a request library file was built. One or more mappings failed. This

message appears with other messages that provide more information about the error.

User Action: Use the /LOG qualifier on the BUILD, CREATE, or MODIFY commands to determine which mappings failed.

NORLBEXT **no requests specified for this library**

Severity: Error (E)

Explanation: A mapping message. The request library definition contains no REQUEST IS instructions specifying the names of the requests to be placed in the request library file or it contains no FORM IS instructions.

User Action: Include REQUEST IS or FORM IS instructions in the request library definition, naming the requests or forms to be built into the request library file.

NOSEMICLN **missing <;> at end of instruction**

Severity: Error (E)

Explanation: A syntax message. A semicolon is missing at the end of an instruction.

User Action: Insert a semicolon at the end of the instruction.

NOSUCHFLD **no such field**

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

Explanation: A mapping message. A mapping instruction or a CONTROL FIELD IS instruction references a field that is not on the active form or in a record used in the request.

User Action: Make sure that the field name in the mapping instruction is correct and that the field is in a record or on the active form used in the request.

NOTINCTL **not within the scope of a control field**

Severity: Error (E)

Explanation: A mapping message. You tried to perform some action that is valid only within a control field (for example, referencing dependent names).

User Action: If the action is not within a control field, delete the conditional instruction syntax, or perform the action within a control field.

OUTSTRTRU **output string truncated**

Severity: Explicit: Error (E) Input or return mappings
Warning (W) Output mappings

%ALL: Information (I)

Explanation: A mapping message. A data type conversion error occurred due to length of character-coded Text data or Numeric Byte Data String (NBDS) when you mapped a literal string to a form or record field. See the documentation on LIB$CVT_DX_DX in the *VAX/VMS Run-Time Library Routines Reference Manual.*

User Action: Make the source and destination field lengths and sizes compatible.

POSINVDAT **mapping may introduce invalid data into form field**

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

Explanation: A mapping message. An output mapping may introduce invalid data into the form field during data type conversion. This warning occurs when the record field is of data type TEXT (T or VT) and the form field is of data type NUMERIC MIX (NX) or UNSIGNED NUMERIC (NU).

User Action: Check to make sure that the data in the record field will map correctly to the form field.

PRKNOACTS **PROGRAM KEY definition does not specify any actions**

Severity: Error (E)

Explanation: A mapping message. A PROGRAM KEY IS instruction contains no OUTPUT, RETURN, or MESSAGE LINE IS instruction.

User Action: Include an OUTPUT TO, RETURN TO, or MESSAGE LINE IS instruction in the PROGRAM KEY IS instruction.

PSWDTLONG **illegal CDD password -- longer than 64 characters**

Severity: Error (E)

Explanation: A syntax message. The CDD password in a FORM IS, RECORD IS, or REQUEST IS instruction is longer than 64 characters.

User Action: Reduce the length of the CDD password to 64 characters.

REFFEWSS **field reference specifies fewer dimensions <number> than the field definition <number>**

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

Explanation: A mapping message. The form or record field reference specifies fewer dimensions than are in the form or record definition.

User Action: Adjust the dimensions in the field reference to correspond to the number specified in the form or record definition for the field.

REFINDIM **in dimension <number>**

Severity: Error (E)

Explanation: A mapping message. This message appears with other field reference error messages. It indicates which dimension is in error.

User Action: Correct the dimension in the reference that is in error.

REFLOWLRG **low limit <number> of reference is larger than high limit <number>**

Severity: Error (E)

Explanation: A mapping message. The low limit of the subscript range specified in the array reference is greater than the high limit of the range.

User Action: Adjust the limits in the subscript range reference so that the low limit is less than or equal to the high limit.

REFMANSS **field reference specifies more dimensions <number> than the field definition <number>**

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

Explanation: A mapping message. The form or record field reference specifies more dimensions than are in the form or record definition for the field.

User Action: Adjust the dimensions of the form or record field reference to correspond to the number specified in the form or record definition for the field.

REFMAXDIM **reference specifies more than 255 dimensions for a field**

Severity: Error (E)

Explanation: A mapping message. A field reference specifies more than 255 dimensions.

User Action: Specify fewer than 255 dimensions for the field reference.

REFSSOUT **reference subscripts <number to number> out of range of field subscripts <number to number>**

Severity: Error (E)

Explanation: A mapping message. The field reference specifies subscript limits that are out of range of the lower and upper bound of the record or form field referenced.

User Action: Adjust the subscripts of the sending or receiving field reference so they are within range of the field definition.

SRCLENLEQ **SOURCE LENGTH must be less than or equal to <number>**

Severity: Explicit: Error (E) Input or return mappings
Warning (W) Output mappings

%ALL: Information (I)

Explanation: A mapping message. A data type conversion error occurred because the length (including scale factor) of the sending field is incompatible with the data type of the receiving field.

User Action: Change the length of the sending field (including scale factor) to be less than or equal to the length specified in the message.

SRCMORDIM **source field has more dimensions than destination**

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

Explanation: A mapping and syntax message. The sending field of an OUTPUT, INPUT, or RETURN mapping instruction has more dimensions than the receiving field.

User Action: Make the number of dimensions in the sending field of the mapping instruction the same number as in the receiving field.

SRCMORENT **source field has more entries than destination**

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

Explanation: A mapping and syntax message. The sending field reference contains a greater number of elements than the receiving field reference.

User Action: Adjust the sending field reference to contain the same number of elements as the receiving field reference.

STRMIS **stride value for dimension is missing - dimension has variable stride**

Severity: Error (E)

Explanation: A mapping message. The stride value for a dimension in a record field definition is missing. The stride is the difference between the addresses of successive elements in the same dimension of an array.

User Action: RDU can not process record definitions that have arrays with variable stride. Add a stride to the field definition.

SYNTAXERR **found <syntax-error> when expecting <correct-syntax>**

Severity: Error (E)

Explanation: A syntax message. The message identifies the error and the syntax that was expected.

User Action: Correct the syntax error.

TOOMANACT **too many active forms specified in request**

Severity: Error (E)

Explanation: A mapping message. You used more than one USE FORM or DISPLAY FORM in a base or conditional portion of a request. TDMS will attempt to display more than one form during a single call to a request. Only one form can be active during any single request call, or at any one level in the request.

User Action: Have only one USE FORM or DISPLAY FORM instruction that is active during any single request call.

TOOMNYBEL **number of bells cannot be greater than 255**

Severity: Error (E)

Explanation: A syntax message. The number of bells in a RING BELL instruction exceeds 255.

User Action: Change the number of bells to between 0 and 255.

UNTERMDESC **unterminated description text**

Severity: Error (E)

Explanation: A syntax message. An END-OF-FILE has been reached before a DESCRIPTION instruction has been terminated with a */.

User Action: End the DESCRIPTION instruction with a */.

UNTERMPASS **unterminated CDD password**

Severity: Error (E)

Explanation: A syntax message. The password on a path name in a FORM IS, RECORD IS, or REQUEST IS instruction is not terminated with a right parenthesis before the end of the line.

User Action: Terminate the password in the FORM IS, RECORD IS, or REQUEST IS instruction with a right parenthesis before the end-of-line is reached.

UNTERMSTRG **unterminated quoted string - assuming string termination at end of line**

Severity: Error (E)

Explanation: A syntax message. You specified a quoted string that was longer than a single line.

User Action: Terminate the quoted string before the end of the line.

UPPBNDMIS **upper bound for dimension is missing - dimension has variable upper bound**

Severity: Error (E)

Explanation: A mapping message. The upper bound of a record field definition is missing.

User Action: RDU cannot process record definitions that have arrays with variable upper bounds. Add the upper bound to the field reference.

TDMS Run-Time Error Messages C

This appendix lists all TDMS run-time error message codes alphabetically by mnemonic. With each code there is an explanation of the severity of the error, an explanation of the message, and recommended user action.

TDMS returns run-time error messages during the execution of a program. If you use TSS$SIGNAL to signal an error from a TDMS call, TDMS also reports extended status (where applicable) from other subsystems.

The error message format is:

%TSSFDV-N-mnemonic, message

TSS — Is the facility name.

N — Is a letter indicating the severity of the error, as follows:

- S — Successful completion.
- I — Information only.
- W — Warning.
- E — Error.
- F — Fatal severe error. Execution of the program does not continue and the procedure does not produce any output.

mnemonic Is a three- to nine- character string that identifies the error.

message Identifies the error.

Note that if there is a problem with the TDMS software, a call might return the following error:

```
%SYSTEM-F-ACCVIO, access violation, reason mask=XX,
virtual address=XXXXXXXX, PC=XXXXXXXX, PSL=XXXXXXXX
```

X can be any hexadecimal number. For recommended user action and explanation, see BUGCHECK.

BUGCHECK **fatal internal software error**

Severity: Severe (F)

Explanation: An unexpected, fatal internal software error occurred in VAX TDMS. This usually means a bug in the TDMS software.

User Action: Collect as much information as possible and submit a Software Performance Report (SPR). To do so, use TSS$SIGNAL to report extended status. See the *VAX TDMS Release Notes* for information on what additional data should be submitted with a VAX TDMS SPR.

CANINPROG **cancel in progress on channel**

Severity: Severe (F)

Explanation: A TSS$CANCEL was issued on the channel while a call that is being cancelled was in progress, and the cancel operation is still underway.

User Action: Check your program to be sure that:

- TSS$CANCEL has returned to the program.
- The call that was cancelled by TSS$CANCEL has returned to the program.

A cancel is complete *only when* the cancelled call returns to the program. You can add source code to set a flag in the program when a call returns TSS$_CANCEL to indicate the cancel is complete. You can then issue another TDMS call.

CONVERR **data type conversion error**

Severity: Severe (F)

Explanation: A run-time data type conversion error occurred during the execution of a request. The reasons why this may occur are:

1. The source field length is greater than the destination field length on an output mapping. In this and similar instances, RDU builds request library files with warnings that certain mappings may cause a data type conversion error.
2. The record parameter(s) on the TSS$REQUEST call is incorrect because:
   - Something other than the record was passed as a parameter
   - The order of the records that were passed as parameters is not the same order as the records specified in the RECORD IS instruction in the request
3. The CDD record definition referenced in the request changed and either the request library file was rebuilt or the program was recompiled. As a result, the record definition in the request library file is not the same as the record definition in the program.

User Action: Use TSS$SIGNAL to report extended status. The extended status includes which field is in error and the specific conversion error that occurred (for example, access violation). Also, use the Trace Facility to see what mapping instruction is in error. The trace log shows the list of successful mappings. Depending on the reason for the error as listed under Explanation, take the following action:

1. Correct the request or the program logic to change the value of the field that is in error. Or, increase the range or size of the field in error.
2. Examine the record(s) passed on the TSS$REQUEST call to see if they are the correct addresses of the records, and are in the same order as the RECORD IS instructions in the request.
3. Make sure that the request library file and the program use the same CDD record definition. You may have to rebuild the request library file or recompile and relink the program. Check:
   - The CDD record definition
   - The request library file
   - The program

If you still get a data type conversion error when you are sure that you are passing the record parameter(s) correctly and that the request library file and application program were both built against the same CDD record definition, submit an SPR.

DISFORERR **display form failed**

Severity: Severe (F)

Explanation: A request was executed that included an instruction to display a 132-column form of more than 13 lines on a VT100-compatible terminal that does not have the Advanced Video option (AVO).

User Action: Use a terminal with AVO to execute the request or change the form. If you change the form, you must rebuild the request library file. 132-column forms with more than 13 lines require the Advanced Video feature.

ERROPNDEV **error opening device**

Severity: Severe (F)

Explanation: An error occurred opening the device specified on a TSS$OPEN call.

User Action: Use TSS$SIGNAL to report extended status and see what specific error occurred when you tried to open the device (for example, device already allocated). Check the device name parameter on the TSS$OPEN call; correct the source program.

ERROPNRLB **error opening request library**

Severity: Severe (F)

Explanation: An error occurred opening the request library file on a TSS$OPEN_RLB call.

User Action: Use TSS$SIGNAL to report extended status and see what specific error occurred when trying to open the request library file (for example, file not found). Check the request library file parameter on the TSS$OPEN_RLB call; correct the source program.

ERROPNTRA **error opening trace log file**

Severity: Severe (F)

Explanation: An error occurred opening the trace log file during a TSS$TRACE_ON call.

User Action: Use TSS$SIGNAL to report extended status and see what specific error occurred when trying to open the file. Also, verify that the trace log file logical name is correctly defined (issue the command SHOW LOGICAL TSS$TRACE_OUTPUT). If it is not defined, check the translation of the default trace log file logical name, DBG$OUTPUT.

ILLDEVCHAR **illegal device characteristics**

Severity: Severe (F)

Explanation: For TSS$OPEN, the device specified is not a valid terminal device that TDMS can open. For TSS$REQUEST this error occurs if you are running the application on a VT52 terminal and the form specified in the DISPLAY FORM instruction in the request contains one or both of the following features:

- 132-column form
- Scrolled region

User Action: Check the device name parameter if the error occurred on the TSS$OPEN call. Correct the source program to open a valid terminal device. Check the form definition using the FDU LIST FORM command if the error occurred on the TSS$REQUEST call. You should either change the form definition or run the application on a VT100 compatible terminal. Note that if you change the form definition, you must rebuild the request library file.

INSVIRMEM **insufficient virtual memory**

Severity: Severe (F)

Explanation: An attempt to allocate dynamic storage during the execution of a TDMS call failed.

User Action: Check the following:

1. The virtual memory usage by the application program. You can do this by:

   - Linking your program with the /DEBUG qualifiers and with SYS$LIBRARY:STARLET.
   - Running the program and issuing the following command at the DBG> prompt:

     ```
     CALL LIB$SHOW_VM
     ```

   This call shows the number of bytes of virtual memory still allocated. You can also issue this comand before and after a TDMS call to be sure you have

enough virtual memory available for the TDMS call. Also step through the application program and be sure the program or its subprograms or subroutines are deallocating virtual memory when the subroutine or subprogram returns to the calling program. In the debugger, you can set breakpoints at LIB$GET_VM; when you reach a breakpoint, you can issue the command SHOW CALLS to determine the caller of LIB$GET_VM.

2. The VIRTUALPAGECNT SYSGEN parameter. If it is too small, see your system manager.

If you still get this error message, it may be a TDMS bug. You should submit an SPR.

INVARG **invalid arguments**

Severity: Severe (F)

Explanation: One or more of the parameters passed to a TDMS call is incorrect or you passed too many or too few arguments on the call. Check the individual call in the reference section to determine the correct way to pass the parameters.

User Action: Verify and fix the parameters for the call (check the order of parameters passed, the passing mechanism for each parameter, and the number of parameters for the call). Correct the source program.

INVCHN **invalid channel**

Severity: Severe (F)

Explanation: An invalid channel number was passed. The channel parameter must be the channel number returned by the TSS$OPEN call.

User Action: Check the channel parameter on the call that failed and correct the source program.

INVDSC **invalid descriptor**

Severity: Severe (F)

Explanation: The descriptor used to pass a string parameter to a TDMS call is invalid. Usually, this means that your parameters are out of order or you have passed something by reference that should be passed by descriptor.

User Action: Check all parameters to the TDMS call that should be passed by descriptor to make sure that they are being passed correctly. Correct the source program.

INVRLBID **invalid request library id**

Severity: Severe (F)

Explanation: The library-id parameter on a TSS$REQUEST or TSS$CLOSE_RLB call was invalid. The request library id must be a number returned by a successful TSS$OPEN_RLB call.

User Action: Verify that the request library id passed was the same number returned by the TSS$OPEN_RLB call. Also check the order of arguments passed and the passing mechanism for each argument. Correct the source program.

INVTXTLEN **invalid text length**

Severity: Severe (F)

Explanation: For TSS$READ_MSG_LINE, the message prompt is too long (it must be less than 80 or 132 characters, depending on whether an 80 or 132-column form is displayed). For TSS$WRITE_MSG_LINE, the message text is too long (it must conform to the same restrictions as the message prompt).

User Action: Check the message prompt parameter if the error occurred on the read message line call. Check the message text parameter if the error occurred on the write message line call. Correct the source program.

MULTFORM **multiple active forms are illegal**

Severity: Severe (F)

Explanation: More than one DISPLAY FORM or USE FORM instruction was encountered during the execution of a conditional request. For example, if two parts of a conditional request are evaluated as true, and each part contains a DISPLAY FORM or USE FORM instruction, this error occurs.

User Action: Use the Trace Facility to check the execution of request, including control field values. Correct the request and/or the program logic so that only one DISPLAY FORM or USE FORM instruction is executed after control field evaluation.

NOIO **no I/O in progress on channel**

Severity: Informational (I)

Explanation: The caller attempted to cancel I/O on a TDMS channel, but there was no call in progress.

User Action: Check your program logic to make sure that you should be issuing the TSS$CANCEL call at the time you are doing so.

NONPRICHA **field contains non-printable characters**

Severity: Severe (F)

Explanation: A TDMS call attempted to output non-printable characters. This error can occur on three calls:

1. The message prompt on the TSS$READ_MSG_LINE call
2. The message text on the TSS$WRITE_MSG_LINE call
3. An output instruction in a request on the TSS$REQUEST call

User Action: Depending on the reason for the error as listed under Explanation, you can:

1. Check the prompt string on TSS$READ_MSG_LINE to be sure you are passing printable characters.
2. Check the message line text on TSS$WRITE_MSG_LINE to be sure you are passing printable characters.
3. Use TSS$SIGNAL to report extended status following the TSS$REQUEST call to see the form field that is in error. Usually, you can correct this problem by initializing the record fields before a call to TSS$REQUEST.

In all cases, correct the source program to pass a valid string.

NORMAL **normal successful completion**

Severity: Success (S)

Explanation: The TDMS call completed successfully.

User Action: None. This message is informational.

PRKCHECK **request was terminated by a check PRK**

Severity: Informational (I)

Explanation: The operator terminated the request input by using a program request key defined in the request. The PROGRAM KEY IS instruction defined the PRK as CHECK.

User Action: None. This is an informational return status to notify the program that the PRK was used by the operator and that the instruction(s) contained within the PROGRAM KEY IS instruction were executed. Note: a CHECK PRK means that you can be sure only that input-required form fields were completed by the operator.

PRKNOCHECK **request was terminated by a nocheck PRK**

Severity: Informational (I)

Explanation: The operator terminated the request input by using a program request key defined in the request. The PROGRAM KEY IS instruction defined the PRK as NOCHECK.

User Action: None. This is an informational return status to notify the program that the PRK was used by the operator and that the RETURN instruction(s) contained within the PROGRAM KEY IS instruction were executed.

NOTE: A NOCHECK PRK means that the request input is terminated without any check on whether or not form fields mapped for input were completed. Therefore, no data is returned to the program buffer except a value returned from a RETURN instruction associated with the PRK.

REQNOTFOU **request not found**

Severity: Severe (F)

Explanation: The request specified by the request-name parameter on the TSS$REQUEST call was not found in the request library file specified by the library-id parameter.

User Action: Use the Trace Facility to see which request name was passed as a parameter on the TSS$REQUEST call. Check that request name with the request library definition in the CDD to be sure that the request name you are passing is included in the request library file. If you cannot find the request name in the request library definition, the request is not included in the request library file.

RLBOBS **RLB is obsolete, rebuild**

Severity: Severe (F)

Explanation: The request library file you are using is no longer valid.

User Action: Rebuild the request library file.

SYNASTLVL	**synchronous calls may not be called at AST level**
Severity:	Severe (F)
Explanation:	A TDMS call was made from an AST routine (a routine running at AST level).
User Action:	TDMS calls cannot be issued from an AST routine. Correct the source program.
TRAOFF	**trace already off**
Severity:	Informational (I)
Explanation:	A call to TSS$TRACE_OFF was made when the Trace Facility was already turned off.
User Action:	None. This is an informational return status only. You can check your source program to see why trace was already off when you made your call to TSS$TRACE_OFF.
TRAON	**trace already on**
Severity:	Informational (I)
Explanation:	A call to TSS$TRACE_ON was made when the Trace Facility was already turned on. For example, you see this message if the logical TSS$TRACE_OUTPUT was defined before the application program was run and the program issued its own TSS$TRACE_ON call.
User Action:	None. This is an informational return status only. You can check your source program and/or the definition of TSS$TRACE_OUTPUT to see why Trace was already on.

TDMS/DATATRIEVE Error Messages D

This appendix lists the TDMS error messages that can be issued when a VAX DATATRIEVE application uses TDMS. After each message, there is an explanation of the severity of the error, an explanation of the message, and a recommendation of the action the user should take to correct the problem.

The error message format is:

%TSSFDV-N-mnemonic, message

TSSFDV — Is the facility name.

N — Is a letter indicating the severity of the error, as follows:

S	Successful completion
I	Information only
W	Warning
E	Error
F	Fatal severe error

mnemonic — Is a three- to nine- character string that identifies the error.

message — Identifies the error.

FLD **invalid field specification**

Severity: Error (E)

Explanation: TDMS cannot find the specified field name on the form.

User Action: Make sure that the form and field names specified to DATATRIEVE are correct.

FRM **invalid form description**

Severity: Error (E)

Explanation: TDMS cannot find the specified form in the .RLB file. This message usually appears when the form has been copied from one CDD location to another (either by an FDU COPY FORM command, a DMU COPY command, or a DMU BACKUP command) without being modified. This message can also be generaged when the form was created from an FMS form that had a different form name.

User Action: Run FDU to modify the form. Note that no changes need to be made to the form. Simply enter the form editor and save the form in the CDD to solve this problem.

IOL **I/O error on form library**

Severity: Error (E)

Explanation: The DATATRIEVE application is trying to get a form from an .RLB file, but TDMS cannot open that .RLB file. This problem can occur if you specify the incorrect .RLB file, or if you are trying to reference an .RLB file with a version of DATATRIEVE that has not been linked with TDMS.

User Action: Make sure the .RLB file specified is correct. Also make sure that DATATRIEVE has been linked with TDMS.

IOR **terminal I/O error**

Severity: Error (E)

Explanation: TDMS has issued a call to the $QIO system service and the QIO has failed. This problem can occur when TDMS initializes the terminal, does input or output to the terminal, or completes operations to the terminal.

User Action: Re-try the DATATRIEVE operation.

ITT **invalid terminal type**

Severity: Error (E)

Explanation: TDMS has found that the terminal type is not supported or that a feature is not supported on the terminal. For example, VT52 terminals do not support scrolled regions and cannot display forms wider than 80 columns.

User Action: Make sure the terminal is a supported terminal type. Refer to the TDMS SPD for a complete list of the supported terminal. Also make sure that the terminal is set to the appropriate type. You can check the terminal type by issuing the SHOW TERMINAL command at the DCL prompt. If the terminal is not set to the appropriate type, issue the SET TERMINAL/INQUIRE command at the DCL prompt.

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.

A

B

C

D

E

F

M

N

O

P

Q

R

S

T

U

V

W

How to Order Additional Documentation

If you live in:	Call:	or Write:
New Hampshire, Alaska	603-884-6660	Digital Equipment Corp. P.O. Box CS2008 Nashua, NH 03061-2698
Continental USA, Puerto Rico, Hawaii	1-800-258-1710	Same as above.
Canada (Ottawa-Hull)	613-234-7726	Digital Equipment Corp. 940 Belfast Road Ottawa, Ontario K1G 4C2 Attn: P&SG Business Manager or approved distributor
Canada (British Columbia)	1-800-267-6146	Same as above.
Canada (All other)	112-800-267-6146	Same as above.
All other areas	—	Digital Equipment Corp. Peripherals & Supplies Centers P&SG Business Manager c/o DIGITAL's local subsidiary

Note: Place prepaid orders from Puerto Rico with the local DIGITAL subsidiary (phone 809-754-7575).

Place internal orders with the Software Distribution Center, Digital Drive, Westminster, MA 01473-0471.

VAX TDMS
Reference Manual
AA-HU17A-TE

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