Update Notice #1

February 1987

VAX TDMS Request and Programming Manual

AD-GS14B-T1

NEW AND CHANGED INFORMATION

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

INSTRUCTIONS

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

Old Page	New Page
Title Page/Copyright	Title Page/Copyright
iii to xi	iii to xi
xvii/xviii	xvii/Blank
1-11/1-12	1-11/1-12
3-1/3-2	3-1/3-2
6-11/6-12	6-11/6-12
Chapter 11	Chapter 11
Chapter 18	Chapter 18
Index	Index
Reader's Comments/Mailer	Reader's Comments/Mailer

VAX TDMS Request and Programming Manual

Order No. AA-GS14B-TE
Including AD-GS14B-T1

February 1987

This manual describes the TDMS Requestion Definition Utility (RDU). It explains how to create TDMS requests and invoke them from application programs.

OPERATING SYSTEM:	**VMS**
	MicroVMS
SOFTWARE VERSION:	**VAX TDMS V1.7**

digital equipment corporation, maynard, massachusetts

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

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

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

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

The following are trademarks of Digital Equipment Corporation:

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

digital™

Contents

How to Use This Manual xiii

Technical Changes and New Features xvii

Part One: Creating Requests

Introduction to Requests

1.1 General Concepts of Requests 1-1
1.1.1 Using Form Definitions 1-2
1.1.2 Using Record Definitions 1-2
1.2 Using Request Instructions. 1-3
1.2.1 Using the FORM IS Instruction 1-4
1.2.2 Using the RECORD IS Instruction. 1-5
1.2.3 Using the DESCRIPTION Instruction and Comment Text . . . 1-6
1.2.4 Using the CLEAR SCREEN Instruction 1-6
1.2.5 Displaying the Form 1-6
1.2.5.1 Using the DISPLAY FORM Instruction 1-6
1.2.5.2 Using the USE FORM Instruction 1-7
1.2.6 Moving Data to and from the Form. 1-9
1.2.6.1 Using the INPUT TO Instruction 1-9
1.2.6.2 Using the OUTPUT TO Instruction. 1-11
1.2.6.3 Using the WAIT Instruction 1-12
1.2.7 Using Video Field Instructions 1-12
1.2.8 Using the END DEFINITION Instruction 1-13
1.3 Rules for Entering Request Instructions 1-13

2 Using the Request Definition Utility (RDU)

2.1 How RDU Uses the CDD 2-2
2.1.1 Setting RDU to Your CDD Directory 2-2
2.1.1.1 Defining CDD$DEFAULT. 2-2
2.1.1.2 Using the SET DEFAULT Command 2-3
2.1.2 Using the SHOW DEFAULT Command 2-3
2.1.3 Naming Requests and Specifying Forms and Records 2-3
2.2 Creating a Request. 2-5
2.2.1 Using the Interactive Method. 2-5
2.2.2 Using the File Method 2-6
2.2.3 Using a Command File - DCL or RDU Level 2-6
2.3 Correcting Errors. 2-7

2.4 Validating Requests 2-7
2.4.1 Changing the RDU Validation Option 2-8
2.4.2 Using the /[NO]STORE Qualifier 2-9
2.4.3 How RDU Validates a Request 2-10
2.5 Copying Requests in the CDD 2-10
2.6 Modifying a Request 2-11
2.7 Listing a Request 2-11
2.8 Deleting a Request 2-12
2.9 Exiting RDU 2-13

3 Mapping Between Form Fields and Record Fields

3.1 How the TDMS Mapping Instructions Work 3-2
3.2 How to Specify Fields in a Mapping Instruction 3-3
3.3 When to Use the %ALL Syntax 3-4
3.3.1 Using %ALL to Map an Entire Form 3-4
3.3.2 Using %ALL to Map Between a Form and a Larger Record . . . 3-6
3.3.3 Using %ALL to Map Between a Form and a Smaller Record . . . 3-7
3.4 When to Use Explicit Mapping Syntax 3-9
3.4.1 Explicitly Mapping Between a Form and a Record 3-9
3.4.2 Using Explicit Syntax to Map from One Field to Many Fields . . 3-10
3.4.3 Making Explicit References Unique 3-13
3.4.3.1 Using Group Field Names 3-15
3.4.3.2 Using the Record Name 3-17
3.5 Using the %ALL and Explicit Syntax in the Same Request 3-19
3.6 Mapping from a Form to a Group Record Field 3-21

4 Making Sure Your Request Mappings Are Correct

4.1 Rules for Mapping 4-1
4.2 Making Sure Record and Form Field Structures Are Compatible . . . 4-2
4.2.1 TDMS Record Field Structures 4-2
4.2.2 TDMS Form Field Structures 4-3
4.2.3 Compatible Form and Record Field Structures 4-3
4.3 Making Sure Field Data Types and Length Are Compatible 4-4
4.3.1 TDMS Form Field Data Types and Lengths 4-4
4.3.2 The Record Data Types to Which You Map 4-6
4.4 Creating Mappings Between Compatible Data Types 4-7
4.5 Form Definition Listings 4-10
4.5.1 How to List the Form Definition 4-11
4.5.2 What You Need to Know About Form Definitions 4-11
4.6 Record Definition Listings 4-12
4.6.1 How to List the Record Definition 4-12
4.6.2 What You Need to Know About Record Definitions 4-12

5 Finding and Correcting Your Errors

5.1 Syntax Errors Found by RDU 5-1
5.2 Semantic Errors Found by RDU 5-3
5.2.1 Mapping Errors 5-3
5.2.1.1 %ALL Warning and Information Messages 5-3
5.2.1.2 Explicit Mappings and Error and Warning Level Messages 5-4
5.2.2 Other Semantic Errors Found by RDU 5-5
5.3 Semantic Errors Not Found by RDU 5-6
5.3.1 Order Execution Errors 5-6
5.3.2 Mapping Errors 5-7
5.3.3 Form-Related Errors 5-7
5.4 Correcting Errors. 5-8
5.4.1 Using the EDIT Command 5-8
5.4.2 Using the MODIFY Command 5-8
5.4.3 Defining RDU$EDIT. 5-8
5.4.4 Using the SAVE Command 5-9

6 Using Conditional Instructions in Requests

6.1 Using Conditional Instructions. 6-1
6.2 Conditional Requests 6-2
6.3 Using Conditional Requests 6-3
6.4 How TDMS Executes a Conditional Instruction at Run Time. 6-5
6.4.1 Specifying Control Values 6-6
6.4.1.1 Specifying More Than One Conditional Instruction 6-8
6.4.1.2 Using Nested CONTROL FIELD IS Instructions 6-8
6.4.2 Specifying Case Values 6-9
6.4.2.1 Using the NOMATCH Case Value. 6-9
6.4.2.2 Using the ANYMATCH Case Value. 6-10
6.4.2.3 Conditional Use of Forms 6-11
6.4.2.4 Case Values When You Use More Than One Control Value 6-12
6.4.3 Match Instructions in a CONTROL FIELD IS Instruction. . . . 6-12

7 Mapping Between Form Arrays and Record Arrays

7.1 What Is an Array? 7-1
7.1.1 Types of Form Arrays You Can Map to and from. 7-2
7.1.2 Types of Record Arrays You Can Map to and from 7-4
7.2 Syntax for Mapping Between Form and Record Arrays 7-6
7.2.1 Explicit Syntax for Mapping Array Elements 7-7
7.2.2 %ALL Syntax for Mapping Array Elements 7-8

7.3 Rules for Mapping Arrays 7-9
7.3.1 Explicit Mappings and Errors 7-10
7.3.2 %ALL Mappings and Errors. 7-10
7.4 Examples of Mapping Indexed and Scrolled Arrays 7-11
7.4.1 Explicit Mapping of Scrolled or Indexed Arrays 7-12
7.4.2 %ALL Mappings 7-14
7.4.2.1 %ALL Mapping and a Scrolled Array 7-14
7.4.2.2 Using %ALL Mappings and Indexed Arrays 7-15
7.4.3 Explicitly Mapping a Subset of a Scrolled or Indexed Array . . . 7-17
7.4.4 Mapping Scrolled Arrays to Several Record Arrays 7-18
7.4.4.1 Explicitly Mapping Several Scrolled Arrays 7-18
7.4.4.2 %ALL Mapping of Several Scrolled Arrays 7-20

8 Advanced Mapping Between Arrays

8.1 Horizontally-Indexed Scrolled Form Arrays 8-1
8.2 Two-Dimensional Record Arrays. 8-2
8.3 Syntax for Mapping Two-Dimensional Arrays. 8-4
8.4 General Rules for Two-Dimensional Arrays 8-4
8.5 Examples of Mapping Two-Dimensional Arrays 8-5
8.5.1 Explicit Syntax to Map a Two-Dimensional Array 8-5
8.5.2 Rules for %ALL Mapping of Two-Dimensional Arrays 8-7
8.5.3 %ALL to Map a Two-Dimensional Array 8-7
8.5.4 Mapping a Subset of a Two-Dimensional Array. 8-9

9 Using an Array as a Control Value

9.1 How to Use an Array As a Control Value to Collect Varying Elements 9-1
9.2 How TDMS Evaluates a Control Value Array at Run Time 9-2
9.3 Rules for Specifying the Control Value Array 9-4
9.3.1 Explicitly Assigning Values to %LINE and %ENTRY 9-6
9.3.2 Using a Work Array as a Control Value Array 9-6
9.3.3 Specifying an Entire Array as a Control Value 9-6
9.4 Example - Using a One-Dimensional Control Value Array 9-7
9.5 Example - Using a Two-Dimensional Control Value Array 9-10

10 How to Display and Input Data in a Scrolled Region

10.1 How TDMS Displays and Collects Data in a Scrolled Array 10-2
10.2 How to Allow the Operator to View Data in Scrolled Regions 10-5

11 Key Definition Instructions

11.1 Key Definition Instructions . . . 11-1
11.2 Which Keys Can Be Specified . . . 11-2
11.3 Rules of Precedence . . . 11-5
11.4 Defining Program Request Keys (PRKs) . . . 11-5
11.4.1 Using Program Request Keys . . . 11-7
11.4.2 Creating a Request That Uses a Program Request Key . . . 11-7
11.4.2.1 Default CHECK Mode Modifier . . . 11-8
11.4.2.2 NO CHECK Modifier . . . 11-10
11.4.3 Examples of Using Program Request Keys . . . 11-11
11.4.3.1 Using PRKs to Allow the Operator to Control Application Flow . . . 11-11
11.4.3.2 Using a PRK to Return a Value to a Control Value . . . 11-12
11.5 Using Function Definition Keys . . . 11-14
11.5.1 Key Functions . . . 11-14
11.5.2 Using the ERROR Function to Deassign a Key Definition . . . 11-16
11.5.3 Examples of Key Definitions . . . 11-17
11.5.3.1 Redefining the RETURN Key to Have the NEXT Function . . . 11-17
11.5.3.2 Redefining the F20 Key to Have the REFRESH Function . . . 11-17

Part Two: Creating Request Libraries

12 Working with Request Libraries

12.1 Creating a Request Library Definition . . . 12-1
12.2 Copying a Request Library Definition . . . 12-2
12.3 Listing a Request Library Definition . . . 12-2
12.4 Modifying a Request Library Definition . . . 12-3
12.5 Validating a Request Library Definition . . . 12-3
12.6 Deleting a Request Library Definition . . . 12-4
12.7 Building a Request Library File . . . 12-4

Part Three: Writing Application Programs

13 Introduction to TDMS Programming

13.1 TDMS Programming Calls . . . 13-1
13.2 General Format of TDMS Programming Calls . . . 13-2

14 Using the Primary TDMS Synchronous Calls

14.1 Opening a Request Library File - TSS$OPEN_RLB 14-2
14.2 Opening a Channel - TSS$OPEN. 14-3
14.3 Transferring Data and Displaying Forms - TSS$REQUEST 14-3
14.4 Closing the Request Libary File - TSS$CLOSE_RLB 14-5
14.5 Closing a Channel - TSS$CLOSE 14-6
14.6 Testing the Return Status Code 14-6
14.7 Compiling and Linking a TDMS Program 14-9
14.7.1 Compiling a TDMS Program. 14-9
14.7.2 Linking a TDMS Program 14-9
14.8 Two Simple TDMS Programs. 14-10

15 Using Supplementary Calls

15.1 Reading Messages from the Terminal - TSS$READ_MSG_LINE . . 15-1
15.2 Sending Messages to the Terminal. 15-2
15.2.1 Writing to the Message Line - TSS$WRITE_MSG_LINE . . . 15-3
15.2.2 Interrupting a Request or an Existing Message Line Operation - TSS$WRITE_BRKTHRU 15-3
15.3 Copying the Current Form to a Specific Device - TSS$COPY_SCREEN 15-4
15.4 Canceling Input/Output Calls in Progress - TSS$CANCEL 15-6

16 Using Record Definitions

16.1 Using CDD Record Definitions in BASIC programs 16-2
16.1.1 Referring to a CDD Record Definition in BASIC 16-2
16.1.2 Referring to a CDD Record Definition Containing the VARIANTS Syntax. 16-4
16.1.3 Referring to CDD Array Record Definitions in BASIC 16-5
16.2 Using COBOL to Refer to CDD Record Definitions. 16-7
16.2.1 Referring to a CDD Record Definition in COBOL 16-8
16.2.2 Referring to a CDD Record Definition Containing the VARIANTS Syntax. 16-10
16.2.3 Referring to CDD Array Record Definitions in COBOL. 16-11
16.3 Using FORTRAN to Refer to CDD Record Definitions. 16-13
16.3.1 Referring to a CDD Record Definition in FORTRAN 16-14
16.3.2 Referring to a CDD Record Definition Containing the VARIANTS Syntax. 16-15
16.3.3 Referring to CDD Array Record Definitions in FORTRAN. . .16-16

16.4 Using Record Definitions Created by Database Management Systems 16-18
16.4.1 Using Record Definitions Created by VAX Rdb/VMS 16-19
16.4.2 Using Record Definitions Created by VAX DBMS 16-20
16.4.3 Displaying and Updating Database Records in Scrolled Regions 16-22
16.4.3.1 Defining an Array Record 16-23
16.4.3.2 Declaring the Array Record in the Application and TDMS Requests 16-24
16.4.3.3 Creating a Collection of Records and Loading the Array . 16-24
16.4.3.4 Passing the Array to the Request. 16-25
16.5 Summary of Supported Data Types for Different Languages 16-25

17 Debugging a TDMS Application Program

17.1 How to Enable the Trace Facility 17-1
17.1.1 Defining a Logical Name 17-2
17.1.2 Issuing Trace Calls from an Application Program 17-2
17.2 Results of Using Trace 17-3
17.3 Debugging an Application Using Two Terminals 17-6

18 Application Function Keys (AFKS)

18.1 What Are Application Function Keys?. 18-1
18.2 When Do You Use Application Function Keys? 18-1
18.3 Declaring Application Function Keys 18-2
18.3.1 Terminal Keys You Can Declare as AFKs 18-3
18.3.2 How to Write an AST Routine 18-4
18.4 Removing an AFK Key Definition 18-4

19 Using Asynchronous Calls

19.1 What Are Asynchronous Calls?. 19-1
19.2 When Do You Use Asynchronous Calls? 19-2
19.3 The General Format for Asynchronous Calls. 19-2

Index

Figures

1-1 Request Format. 1-4
1-2 Mapping Between a Record Definition and a Form Definition 1-10
2-1 Suggested TDMS Design Sequence: Effects of Validation 2-9
4-1 Record Field Structures 4-2
4-2 Form Field Structures 4-3
6-1 A Conditional Request. 6-3

6-2 Request Containing a CONTROL FIELD IS Instruction 6-4
6-3 How a Conditional Request Works. 6-7
7-1 Definition of an Array 7-2
7-2 Indexed Array. 7-3
7-3 Scrolled Array. 7-4
7-4 Simple One-Dimensional Record Arrays 7-5
7-5 One-Dimensional Group Arrays 7-5
7-6 Example of Mapping Arrays 7-6
7-7 The Underlying Form Array 7-11
7-8 Explicitly Mapping an Entire Scrolled Form Array. 7-13
7-9 Using %ALL to Map an Entire Scrolled Array. 7-14
7-10 Using %ALL to Map Entire Indexed Arrays 7-16
7-11 Mapping a Subset of an Indexed or Scrolled Array 7-17
7-12 Explicit Mapping of Several Scrolled Arrays. 7-18
7-13 %ALL Mapping of Several Scrolled Arrays. 7-20
8-1 Horizontally-Indexed Scrolled Form Array 8-1
8-2 Using Explicit Syntax to Map a Two-Dimensional Array 8-6
8-3 Using %ALL Syntax to Map Two-Dimensional Arrays 8-8
8-4 Mapping a Subset of a Two-Dimensional Array 8-9
9-1 Using an Array as a Control Value 9-2
9-2 How TDMS Evaluates a Control Value Array at Run Time 9-3
9-3 The Scope of a Dependent Range 9-4
9-4 Illegal Nested Dependent Ranges 9-5
9-5 Specifying Nonarray or Single-Element Control Values 9-5
9-6 Explicitly Assigning Values to %LINE and %ENTRY. 9-6
9-7 Collecting Elements from Several Scrolled Fields 9-7
9-8 Using a Two-Dimensional Array as a Control Value. 9-11
10-1 Displaying Data in a Scrolled Region 10-6
11-1 A Sample PRK Instruction 11-6
11-2 Defining Program Request Keys 11-8
11-3 Using the CHECK Modifier. 11-9
11-4 Using PRKs to Allow Operator Control of Application Flow 11-12
11-5 Using PRKs in Conditional Instructions 11-13
14-1 Request That Displays a Form 14-10
16-1 Referring to a CDD Record Definition in BASIC 16-2
16-2 Referring to a CDD Record Definition Containing the VARIANTS Syntax in BASIC 16-4
16-3 Referring to a CDD Record Definition with Nested Arrays 16-5
16-4 Referring to a Two-Dimensional CDD Array Record Definition in BASIC. 16-7
16-5 Referring to a CDD Record Definition in COBOL 16-9

16-6 Referring to a CDD Record Definition with the VARIANTS Syntax in COBOL 16-10
16-7 Referring to a CDD Record Definition with Nested OCCURS Syntax in COBOL 16-11
16-8 Referring a Two-Dimensional CDD Array Record Definition in COBOL 16-12
16-9 Referring to a CDD Record Definition in FORTRAN 16-14
16-10 Referring to a CDD Record Definition Containing the VARIANTS Syntax in FORTRAN 16-15
16-11 CDDL Record Definition with Nested OCCURS Syntax 16-17
16-12 Referring to a Two-Dimensional CDD Array Record Definition in FORTRAN 16-17
17-1 Sample Trace Output. 17-3

Tables

4-1 Effect of Scale Factor on Form Field Data 4-5
4-2 TDMS Form Field Pictures and Form Field Data Types 4-5
4-3 Record Field Data Types 4-7
4-4 Simplified Compatible Input Mappings (Form Fields to Record Fields) 4-8
4-5 Simplified Compatible Output Mappings (Record Fields to Form Fields) 4-9
11-1 TDMS Key Functions 11-15
11-2 Keys and Their Default Functions 11-16
16-1 Data Type Conversion Chart 16-26
18-1 Application Function Key Codes 18-3

Technical Changes and New Features

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

There are two new RDU commands and one new RDU instruction:

- ATTACH command
- SPAWN command
- DEFINE KEY AS instruction

In addition, the PROGRAM KEY IS instruction has been extended to support additional keys.

Information about the DEFINE KEY AS and PROGRAM KEY IS instructions appears in Chapter 11 of this manual.

There are new parameters for the OUTPUT TO and RETURN TO instructions. Both instructions now have a %TOD (time of day) parameter. The RETURN TO instruction now has a %MODIFIED parameter.

For reference information about all these new RDU features, see the *VAX TDMS Reference Manual.*

```
          CITY            TO CITY,
          STATE           TO STATE,
          ZIP_CODE        TO ZIP_CODE,
          SEX             TO SEX,
          BIRTH_DATE      TO BIRTH_DATE;

END DEFINITION;
```

Figure 1-2: Mapping Between a Record Definition and a Form Definition (Cont.)

You can also specify %ALL with the INPUT instruction. The INPUT %ALL instruction causes TDMS to collect data from all the fields on the form that have identically named record fields.

In Figure 1-2, the request EMPLOYEE_SAMPLE_REQUEST can be rewritten as follows using the INPUT %ALL instruction because all the form fields have matching record fields with identical names:

```
FORM IS          EMPLOYEE_ADD_FORM;
RECORD IS        EMPLOYEE_RECORD;

CLEAR SCREEN;
DISPLAY FORM     EMPLOYEE_ADD_FORM;

INPUT %ALL;

END DEFINITION;
```

The INPUT %ALL instruction maps form fields to record fields with identical names (LAST_NAME, FIRST_NAME, MIDDLE_INITIAL, STREET, CITY, STATE, ZIP_CODE, SEX, and BIRTH_DATE).

RDU does not create an input mapping for EMPLOYEE_NUMBER even though you specify %ALL. EMPLOYEE_NUMBER is defined on the form as a Display Only field, which means TDMS does not allow the operator to enter data in the field and RDU does not create an input mapping for that field.

1.2.6.2 Using the OUTPUT TO Instruction -- You can describe what data you want TDMS to move from the program record and display on the form using the OUTPUT TO instruction. The general format of the OUTPUT TO instruction is:

OUTPUT record-field TO form-field;

You can list a series of output phrases after the keyword OUTPUT. Each phrase must be separated by a comma. For example:

```
OUTPUT FIRST_NAME TO FORM_FIELD_1,
       LAST_NAME  TO FORM_FIELD_2;
```

If you want to output data to all the fields on a form (that have record fields with identical names), you can use %ALL. If you use %ALL, TDMS displays data to all those form fields that have identically named record fields. For example, in the following request, the OUTPUT %ALL instruction outputs data in all the form fields listed after the DESCRIPTION instruction.

```
                    .
                    .
                    .

OUTPUT %ALL;

  DESCRIPTION /*
       EMPLOYEE_NUMBER TO EMPLOYEE_NUMBER,
       LAST_NAME       TO LAST_NAME,
       FIRST_NAME      TO FIRST_NAME,
       MIDDLE_INITIAL  TO MIDDLE_INITIAL,
       STREET          TO STREET,
       CITY            TO CITY,
       STATE           TO STATE,
       ZIP_CODE        TO ZIP_CODE,
       SEX             TO SEX,
       BIRTH_DATE      TO BIRTH_DATE   */;
                    .
                    .
                    .
```

1.2.6.3 Using the WAIT Instruction -- If a request performs output mappings only, and does not include an INPUT TO instruction, you must use the WAIT instruction to ensure that the operator sees any messages or data you display on a form before TDMS clears the screen. The form remains on the screen until the operator acknowledges it by pressing the RETURN key, a program request key (PRK), or a termination key. See Chapter 11, for more information on program request keys. See the the WAIT instruction in the *VAX TDMS Reference Manual* for a list of the termination keys. The format of the WAIT instruction is:

WAIT;

1.2.7 Using Video Field Instructions

If you want to highlight certain fields on the form, you can use the video field instructions to specify what video attributes the field has. For example, the UNDERLINE FIELD instruction lets you specify that a field be underlined when the form is displayed. Or you can use %ALL to change the video attributes of all the fields on the form. The general format for this instruction is:

UNDERLINE FIELD form-field;

Mapping Between Form Fields and Record Fields 3

The requests in Figures 1-1 and 1-2 contain mappings between form and record fields, using the INPUT TO and OUTPUT TO instructions. TDMS has a total of three mapping instructions:

- INPUT TO
- OUTPUT TO
- RETURN TO

With these instructions, you can use two different types of syntax:

- **%ALL syntax**, in which TDMS maps all form fields to and from record fields with the same name
- **Explicit syntax**, in which you specify the field names of the form and record fields you are mapping

In this chapter, you see examples of requests that map data between form and record fields using both types of syntax. In addition, this chapter describes:

- How the TDMS mapping instructions work
- How to specify fields in a mapping instruction
- When to use the %ALL mapping syntax
- When to use the explicit mapping syntax
- How to use the %ALL and explicit syntax in the same request
- How to map from a form to a group record field

3.1 How the TDMS Mapping Instructions Work

There are three mapping instructions:

- The INPUT TO instruction maps data from a form field to one or more record fields. You can use the INPUT TO instruction to let the operator enter data into a form field and return that data to the program in the record field when the request completes.

  If a form field is mapped for input but the operator does not enter data in that field, TDMS returns one of the following to the record field:

  - Data output to the form field by the current request
  - Data in the form field from the immediately previous request call (as a result of the USE FORM instruction)
  - Data associated with the form field by a form definition default (if no other data is in the field)

- The OUTPUT TO instruction maps data to one or more form fields. You can use the OUTPUT TO instruction to display data on the form when the request is invoked. The OUTPUT TO instruction uses a quoted string, a record field, or %TOD (the current date or time) as a source of the mapping.
- The RETURN TO instruction maps data to one or more record fields. The RETURN TO instruction is similar to the INPUT TO instruction with the following exceptions:

  - The RETURN TO instruction uses a form field, a quoted string, %TOD (the current system time in 64-bit format), or %MODIFIED (a value indicating whether an operator has modified a field as a source of the mapping).
  - The RETURN TO instruction does not open the field for input from the operator if the source of the mapping is a form field. Instead, it uses the current contents of that field.

  The RETURN TO instruction is very useful for conditionally returning data when the request completes or when the operator presses a predefined program request key (PRK).

The examples in this chapter use the INPUT TO and OUTPUT TO instructions to explain the general rules of field mapping.

```
ANYMATCH:
     OUTPUT EMPLOYEE_NUMBER TO EMPLOYEE_NUMBER,
            SELECTION       TO SELECTION;
     INPUT  EMPLOYEE_NUMBER TO EMPLOYEE_NUMBER,
            SELECTION       TO SELECTION;

END CONTROL FIELD;
```

Note that the form fields EMPLOYEE_NUMBER and SELECTION are input and output each time an error message is displayed on the form. Each time, therefore, that the program detects a run-time error:

1. The application program places an appropriate value indicating the error in the control value
2. The program calls the request
3. TDMS executes the instructions if the control value matches one of the case values
4. The operator can enter a new employee number and a new menu selection if any one of the errors specified in the request occurs

6.4.2.3 Conditional Use of Forms -- It is best not to use the ANYMATCH case value when you want to conditionally reference forms in case values. Instead, repeat all mappings and include a DISPLAY FORM or USE FORM instruction within each case value.

However, if you must use ANYMATCH in this situation, be aware that a special situation exists when you try to conditionally reference forms in case values. RDU cannot know which case values will be executed at run time. When you have a mapping instruction in the ANYMATCH case value and DISPLAY FORM or USE FORM instructions in any other case value instructions, RDU cannot determine what form, (if any), will be active at run time.

In addition, the TDMS run-time system does not retain context about control field case values and associated instructions once the case value and instructions are executed. Therefore, when ANYMATCH instructions are executed at run time, TDMS does not know about any form referenced in any other case value instructions within the control field.

Keep the following points in mind when you conditionally reference forms in CONTROL FIELD IS instructions that use the ANYMATCH case value:

- Put a DISPLAY FORM or USE FORM instruction in all case values except the ANYMATCH

- Put instructions in ANYMATCH that do not reference form fields
- Do not put a DISPLAY FORM or USE FORM instruction in the ANYMATCH case value

If you are not conditionally referencing forms, put the DISPLAY FORM or USE FORM in the base request.

6.4.2.4 Case Values When You Use More Than One Control Value -- If you have more than one CONTROL FIELD IS instruction, you also have more than one series of case values. You can use the same case value twice if it is under a different control field each time you use it.

If you select case values that are meaningful strings, the request will be clearer to the programmer.

6.4.3 Match Instructions in a CONTROL FIELD IS Instruction

Match instructions are request instructions to be executed when a case value matches the control value. You can associate any number of match instructions with a particular case value.

Each match instruction is a request instruction in itself and must therefore be followed by a semicolon (;).

When you create a conditional instruction, you can specify any request instruction following a case value (except the FORM IS and RECORD IS instructions), including instructions to:

- Display forms
- Input, output, and return data
- Change video attributes
- Use key definition instructions (discussed in Chapter 11)

Key Definition Instructions 11

TDMS allows you to define or redefine the functions of various keys on the terminal. This chapter discusses:

- Key definition instructions
- Which keys can be specified
- Rules that determine which key definition instructions take precedence
- How key definition instructions work

11.1 Key Definition Instructions

Key definition instructions are request instructions that allow you to redefine the function of a terminal key. TDMS has two key definition instructions:

- PROGRAM KEY IS

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

- DEFINE KEY AS

  This instruction specifies a function for a key or key sequence.

When a request that contains a key definition instruction executes at run time, the new key function is enabled. The operator can press the redefined key to obtain special actions such as cursor positioning, menu selection, field validation, or request termination.

The two key definition instructions are discussed in detail later in this chapter.

11.2 Which Keys Can Be Specified

With the PROGRAM KEY IS instruction, you supply the prk-key parameter and the instruction mapping(s) you want to occur when the operator presses that key. In addition, pressing a PRK terminates the request.

With the DEFINE KEY AS instruction, you supply the key-name parameter and the function that you want TDMS to perform when the operator presses that key. In most cases, the keys that you can specify for both instructions are the same. The major difference is that, except for the arrow keys, you can specify GOLD key sequences only with the PROGRAM KEY IS instruction.

In defining keys, you should be aware that there are two modes for the numeric keypad keys (specified with the KEYPAD keyword) as well as the ENTER key. The modes are Numeric and Application. RDU cannot determine the keypad mode at run time, so RDU does not check the mode when it validates the request that maps to one of these keys. For information on how to specify the mode you want, see the description of the KEYPAD MODE IS instruction in the *VAX TDMS Reference Manual*.

There are several categories of key designations:

- The KEYPAD keys are those on the numeric keypad at the right edge of the keyboard. There is a numeric keypad on VT100- and VT200-series terminals. Remember that the KEYPAD keyword is *not* enclosed in quotation marks but the remaining part of the key designation is, for example, PROGRAM KEY IS KEYPAD "8".

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

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

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

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

  PF1
  PF2
  PF3
  PF4

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

  You can specify the following F keys:

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

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

  E1
  E2
  E3
  E4
  E5
  E6

- You use these keywords to specify the arrow keys. In addition, you can specify the GOLD keyword with arrow keys.

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

- There are other key designations that you can specify. The keys are:

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

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

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

- You can use the GOLD keyword in combination with character keys only for the PROGRAM KEY IS instruction. The DEFINE KEY AS instruction restricts using GOLD to combinations with arrow keys (for example, GOLD UPARROW).

  When you specify the GOLD keyword with character keys, GOLD is *not* enclosed in quotation marks but the designation of the character key is, for example, GOLD "&" (GOLD ampersand) and GOLD " " (GOLD space). The default GOLD key is the PF1 key on the numeric keypad. (Note that you can use the DEFINE KEY AS instruction to reassign the GOLD function to some other key.)

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

  A-Z
  a-z
  0-9

  You can specify the following characters with GOLD:

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

11.3 Rules of Precedence

Some keys and key sequences can be defined in several ways. Many keys can be specified with both the DEFINE KEY AS and the PROGRAM KEY IS instructions. In some instances such as BACKSPACE (CTRL/H), LINEFEED (CTRL/J), and RETURN (CTRL/M), you can specify the same key as an application function key (AFK) that you can define with a PROGRAM KEY IS and/or DEFINE KEY AS instruction.

In all instances where the same key or key sequence has more than one definition, you need to be aware that TDMS processes only *one* of the key definitions. These are the rules that TDMS uses to resolve multiple definitions:

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

11.4 Defining Program Request Keys (PRKs)

Program request keys (PRKs) are keys that you can define in a request to perform mapping instructions. You define a PRK in a request by naming the key or key sequence and associating mapping instructions with that key. When the TDMS application is running and an operator presses a PRK that you defined, TDMS executes the mapping instructions you associated with that PRK and terminates the request.

For example, you can create a request containing a program request key and associated PRK mapping instructions, as shown in Figure 11-1.

PRK_SAMPLE_REQUEST

```
RECORD IS    EMPLOYEE_SAMP_REC;
FORM IS      EMPLOYEE_SAMP_FORM;
DISPLAY FORM EMPLOYEE_SAMP_FORM;

   OUTPUT BADGE_NO TO BADGE;
   INPUT  NAME     TO NAME_FLD;

    PROGRAM KEY IS GOLD "M"

      RETURN "BACKUP" TO WK_CHANGE_FIELD;
      OUTPUT "RETURNING TO MENU" TO MESSAGE;
    END PROGRAM KEY;

END DEFINITION;
```

Program request key

PRK mapping instructions

Figure 11-1: A Sample PRK Instruction

During a TDMS application, when an operator presses the PRK as defined in the PRK_SAMPLE_REQUEST (the GOLD-M key sequence on the keyboard), the TDMS software:

1. Executes the output and returns mapping instructions associated with the PRK as follows:

    - Returns the text string BACKUP to the record field WK_CHANGE_FIELD

    - Displays the string RETURNING TO MENU in the form field MESSAGE

2. Terminates the request call and returns control to the application program

3. Returns a value to the program indicating that the request was terminated by a PRK.

Note that when an operator presses a PRK at run time, TDMS checks that all form fields defined as Response Required have data entered in them (in the default mode of the PRK instruction). If these fields do not have data entered in them, TDMS ignores the PRK and continues executing the instructions in the request. (See the section in this chapter on the default CHECK modifier and the NO CHECK modifier.)

11.4.1 Using Program Request Keys

PRKs are a convenient way for a TDMS operator to communicate with the application program.

By returning messages (that you predefine in a request) to the program, PRKs permit the operator to send run-time messages to the application program. The program can then respond to the condition identified by the operator.

You can use program request keys in your request to let the operator:

- Select a menu option. (For example, you might have a menu in which the operator selects an option by pressing a particular PRK.)
- Notify the application of a change in the sequence of operations. (For example, the program might continue asking for employee data until the operator presses a PRK to indicate readiness for a new employee form.)
- Notify a TDMS application program to exit.
- Indicate that a particular type of operator error occurred and that the program should take certain corrective action.

You can also use PRKs in conditional instructions. Later in this chapter, you see examples of program request key instructions that are used within a conditional instruction to return values to control values. First, however, you learn how to create a simple request containing a program request key.

11.4.2 Creating a Request That Uses a Program Request Key

The request in Figure 11-2 shows that to use a program request key in a request, you must specify:

1. The instruction keywords, PROGRAM KEY IS
2. A valid prk-key
3. The following instructions:
    - One RETURN quoted-string TO record-field instruction
    - One of either (but not both):
        - A MESSAGE LINE IS instruction
        - An OUTPUT quoted-string TO instruction

4. The end phrase END PROGRAM KEY followed by a semicolon

```
RECORD IS EMPLOYEE_ADD_REC;
            .
            .
            .

   PROGRAM KEY IS GOLD "D"                       ←— 1 & 2  Keywords
                                                           and prk-name
      RETURN "DONE" TO WK_CHECK_FLD;
      OUTPUT "This employee record complete"     ←— 3 PRK instructions
              TO MSG_FLD WITH BOLD;
   END PROGRAM KEY;                              ←— 4 End phrase

 END DEFINITION;
```

Figure 11-2: Defining Program Request Keys

Notice that you must end the PROGRAM KEY IS instruction with the end phrase END PROGRAM KEY and a semicolon. Note also that you do not use a semicolon following the PROGRAM KEY IS prk-key phrase.

RDU automatically puts all PRK key designations in uppercase. So, for example, saying GOLD "A" is the same as GOLD "a". At run time, when the operator presses a PRK key, whether it is uppercase or lowercase, TDMS matches it to the PRK key designation in the request. The PROGRAM KEY IS instruction can occur anywhere within a request (except in the header section). TDMS responds to program request keys only after it has executed all output mappings.

11.4.2.1 Default CHECK Mode Modifier -- When you define a program request key, it has the default modifier, CHECK. (You can assign the NO CHECK modifier to the PRK instruction.) With checking in effect, at run time when an operator presses a PRK:

1. TDMS checks to see that all form fields defined as Response Required (that are also mapped for input) contain data

2. TDMS checks to see that all form fields defined as Must Fill fields are filled

3. TDMS checks to see that all field validators (Choice List, Range List, and Check Digits) are met

4. If conditions 1, 2, and 3 are met, the TDMS software:

   - Executes the PRK instructions and terminates the request
   - Returns data from all the form fields that were mapped for input or return

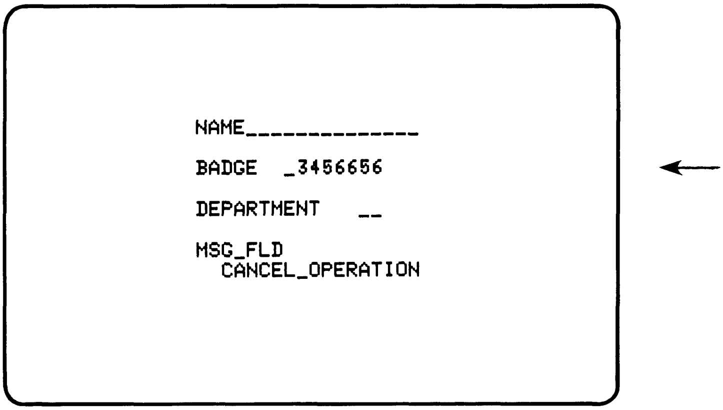
5. If the Response Required fields do not have data in them, the Must Fill fields are not filled, or the field validators are not met, TDMS ignores the PRK

Note that fields on a form that are mapped for input are not necessarily also defined as Response Required fields. When a PRK is pressed, therefore, TDMS might terminate a request even though the operator might not have entered data in all fields mapped for input.

The data returned to a record, therefore, can be any of the following:

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

Figure 11-3 shows how the CHECK modifier works. At run time, when the operator presses the PRK, TDMS checks if the field BADGE has data entered into it by the operator.



```
NAME______________

BADGE  _3456656

DEPARTMENT   __

MSG_FLD
  CANCEL_OPERATION
```

Response Required field (TDMS checks that data is in this field)

(continued on next page)

Figure 11-3: Using the CHECK Modifier

DEPT_INFO_REQUEST

```
FORM IS      DEPT_INFO_FORM;
RECORD IS    DEPT_INFO_RECORD;
DISPLAY FORM DEPT_INFO_FORM;

  INPUT NAME       TO NAME,
        BADGE      TO BADGE,
        DEPARTMENT TO DEPT;

     PROGRAM KEY IS GOLD "C"
                 CHECK;
        RETURN "CANCEL"
                TO  MSG_FLD;
        OUTPUT "CANCEL OPERATION"
                TO  MSG_FLD;
     END PROGRAM KEY;

 END DEFINITION;
```

← Operator presses the GOLD-C key sequence at run time

Figure 11-3: Using the CHECK Modifier (Cont.)

If the BADGE field does have data entered in it, the TDMS software:

1. Outputs the string "CANCEL OPERATION" to the form field MSG_FLD
2. Returns the string "CANCEL" to the record field MSG_FLD
3. Terminates the request and returns the badge number entered by the operator
4. Returns the values that happen to be in the form fields NAME and DEPARTMENT (the operator did not enter values in these fields)
5. Returns a value to the program indicating that the request was terminated by a check PRK and that Response Required fields were checked for input

If the field BADGE does not have data entered in it, the TDMS software:

1. Issues an error message indicating that BADGE is a Response Required field
2. Ignores the PRK and the associated PRK instructions
3. Continues executing the request, including any remaining input instructions

11.4.2.2 NO CHECK Modifier -- If you assign a NO CHECK modifier, at run time TDMS executes only the PRK instructions and terminates the request. That is, when TDMS terminates the request in NO CHECK mode, it executes only the RETURN TO, OUTPUT TO, or MESSAGE LINE IS instructions within the PRK instruction.

TDMS does not:

- Check for Response Required fields, Must Fill fields, or field validators
- Execute any remaining instructions outside the PRK instruction

11.4.3 Examples of Using Program Request Keys

The following sections contain two examples of using PRKs in requests.

11.4.3.1 Using PRKs to Allow the Operator to Control Application Flow -- By defining PRKs that return strings to the program, you give the operator some control over the flow of the application. The request in Figure 11-4 illustrates this concept. The request contains a series of four program request keys. When the operator presses them, TDMS returns a string to signal the program to take one of the following actions:

String	Action
BACK	To go back to a form displayed earlier in an application and redisplay it with information previously entered on that form and to save the information collected so far on this current form.
SKIP	To discard changes entered on this current form and to go to the Menu form in this application so the operator can select a new form.
DONE	To save the data entered so far on this form, to write it to the appropriate record, and to return to the Menu form for another selection.
EXIT	To exit this application and to write all data collected to the appropriate records.

By using these program keys, the operator can:

1. Move among many forms or menu selections in an application (PRK keypad 7 in Figure 11-4)
2. Skip the form that appears on the screen and discard any information entered on this form during this call to the request (PRK keypad 8 in Figure 11-4)
3. Enter information on a form and indicate when he is done and wants the information entered to be written to a file (PRK keypad 9 in Figure 11-4)
4. Exit the application program (PRK keypad 4 in Figure 11-4)

```
CHANGE_EDUCATION_REQUEST

     FORM   IS CHANGE_EDUCATION_FORM;
     RECORD IS EDUC_RECORD;
     RECORD IS CHANGE_WORKSPACE;

     CLEAR SCREEN;
     DISPLAY FORM CHANGE_EDUCATION_FORM;

        KEYPAD MODE IS APPLICATION;
           PROGRAM KEY IS KEYPAD "7"
             CHECK;
             RETURN "BACK" TO WK_CONTROL_FIELD;
           END PROGRAM KEY;

           PROGRAM KEY IS KEYPAD "8"
             NO CHECK;
             RETURN "SKIP" TO WK_CONTROL_FIELD;
           END PROGRAM KEY;

           PROGRAM KEY IS KEYPAD "9"
             CHECK;
             RETURN "DONE" TO WK_CONTROL_FIELD;
           END PROGRAM KEY;

           PROGRAM KEY IS KEYPAD "4"
             CHECK;
             RETURN "EXIT" TO WK_CONTROL_FIELD;
           END PROGRAM KEY;

     OUTPUT  EDUC_UNIVERSITY  TO UNIVERSITY,
             EDUC_START_DATE  TO START,
             EDUC_STOP_DATE   TO STOP,
             EDUC_DEGREE      TO DEGREE;

     INPUT   UNIVERSITY TO EDUC_UNIVERSITY,
             START      TO EDUC_START_DATE,
             STOP       TO EDUC_STOP_DATE,
             DEGREE     TO EDUC_DEGREE;
     END DEFINITION;
```

Figure 11-4: Using PRKs to Allow Operator Control of Application Flow

Note that you must use KEYPAD MODE IS APPLICATION for the application program to recognize data entered on the keypad as a keypad key (rather than numeric data). However, RDU does not check when you create a request that you have specified the KEYPAD MODE IS instruction if you use a keypad key as a PRK.

11.4.3.2 Using a PRK to Return a Value to a Control Value -- You can use a program request key to return a value to a control value in a conditional instruction.

In Figure 11-5, for instance, the operator can press one of two PRKs at run time (keypad 8 or keypad 4) and place predetermined values (MORE or DONE) in the control value ACTION_TO_TAKE. In a subsequent call to this same request, TDMS can evaluate the control value ACTION_TO_TAKE and then execute the appropriate request instructions.

DEPT_LABOR_REQUEST

```
FORM   IS DEPTLABOR_FORM;
RECORD IS DEPTLABOR_WORKSPACE;
RECORD IS LABOR_RECORD;

CLEAR SCREEN;
DISPLAY FORM DEPTLABOR_FORM;

CONTROL FIELD IS ACTION_TO_TAKE

        NOMATCH:
           INPUT   NUMBER  TO LABOR_EMPL_NUMBER,
                   NAME    TO LABOR_NAME,
                   DEPT    TO LABOR_DEPT;

        "MORE":
           OUTPUT
                  LABOR_EMPL_NUMBER    TO NUMBER,
                  LABOR_NAME           TO NAME,
                  LABOR_DEPT           TO DEPT;
END CONTROL FIELD;

INPUT   PROJECTNO TO WK_PROJECT_NO,
        HOURS     TO WK_HOURS,
        CODE      TO WK_OPCODE;

KEYPAD MODE IS APPLICATION;

    PROGRAM KEY IS KEYPAD "8"
        CHECK;
        RETURN "MORE" TO ACTION_TO_TAKE ;
    END PROGRAM KEY;

    PROGRAM KEY IS KEYPAD "4"
        CHECK;
        RETURN "DONE" TO ACTION_TO_TAKE;
    END PROGRAM KEY;
END DEFINITION;
```

Figure 11-5: Using PRKs in Conditional Instructions

Note that, when the request DEPT_LABOR_REQUEST is first called, the control value ACTION_TO_TAKE is blank. TDMS executes the INPUT instructions in the base section and in the NOMATCH case value. It collects basic employee information (NUMBER, NAME, and DEPT) and project information (PROJECTNO, HOURS, and CODE).

The operator can press one of two PRKs:

- The keypad 8 key, indicating there is more information to enter on the same employee.

  TDMS returns the string "MORE" to the control value and terminates the request call. The program calls the DEPT_LABOR_REQUEST a second time. TDMS executes the request instructions following the case value "MORE". It displays the number, name, and department information. In addition, TDMS executes the INPUT TO instruction in the base section of the request and collects additional project-related data on the same employee.

- The keypad 4 key, indicating there is no more information to enter on this employee.

  TDMS returns the string "DONE" to the control value and terminates the request call. The program issues a second call to the request DEPT_LABOR_REQUEST. There is no match between the control field containing "DONE" and the case values. TDMS, therefore, executes the NOMATCH instructions. It collects employee information (NUMBER, NAME, and DEPT) and project data (PROJECTNO, HOURS, and CODE) for a different employee.

11.5 Using Function Definition Keys

Function definition keys are keys that you can define to perform certain TDMS functions whenever an operator presses them. By default, there is at least one key or key sequence for each function. However, you can use the DEFINE KEY AS instruction to reassign those functions to other keys.

The DEFINE KEY AS instruction allows you to change the functions that are associated with keys by default and to assign additional keys to have the same functions. For example, you can define the RETURN key to perform the NEXT function and also have the TAB key retain that function. Instead of having LINEFEED or F13 handle the ERASE function, you can define E3 to have that function and then redefine the LINEFEED or F13 key to the ERROR function.

11.5.1 Key Functions

TDMS has 15 functions that you can assign to keys. Certain keys are assigned these functions by default. All remaining function definition keys are assigned the ERROR function.

The ERROR function in TDMS provides a signal to the operator that the key or key sequence pressed has no executable function. The cursor remains in its same position on the form. In essence, the ERROR function means that there is no function defined for that function definition key.

Table 11-1 lists the TDMS key functions and a description for each of them.

Table 11-1: TDMS Key Functions

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

Table 11-2 shows the function definition keys that are assigned by default to perform the various functions, except the ERROR function. Function definition keys not listed in the table have the ERROR function by default.

Table 11-2: Keys and Their Default Functions

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

Note that CTRL/R and CTRL/W are assigned the REFRESH function by default. Control key sequences can only be enabled by the TSS$DECL_AFK and TSS$DECL_AFK_A programming calls. You cannot specify control key sequences in key definition instructions. However, you can assign the REFRESH function to other keys or key sequences.

11.5.2 Using the ERROR Function to Deassign a Key Definition

In TDMS, all definable keys and key sequences that do not have either a default or specifically defined function are assigned the ERROR function. For example, all the E keys on the editing keypad for VT200-series terminals have ERROR as their default function.

Since all function definition keys must have functions assigned to them, you have to assign the ERROR function to a function definition key that you want to undefine. For example, suppose for VT200-series terminals you want to reassign the ERASE function from the default key (F13) to the Remove key (E3) on the editing keypad. You would need to define both the E3 key and the F13 key:

DEFINE KEY E3 AS ERASE;
DEFINE KEY F13 AS ERROR;

Now when the operator presses E3, the field is erased. When F13 is pressed, the terminal bell sounds indicating that the key has no function.

11.5.3 Examples of Key Definitions

The following examples show how you might use the DEFINE KEY AS instruction.

11.5.3.1 Redefining the RETURN Key to Have the NEXT Function -- By default, TDMS assigns the NEXT function to the TAB key and the DONE function to the ENTER and RETURN keys. The following DEFINE KEY AS instructions exchange those functions so RETURN and ENTER now perform the NEXT function and the TAB key performs the DONE function.

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

Now when the operator wants to move the cursor to the next field, he presses either the RETURN or ENTER key. When he finishes entering data, he presses the TAB key to exit from the request.

11.5.3.2 Redefining the F20 Key to Have the REFRESH Function -- By default, the F20 key on VT200-series terminals is assigned the ERROR function. The following DEFINE KEY AS instruction assigns the REFRESH function to F20.

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

Now, the operator can press CTRL/R, CTRL/W, or F20 to clear and repaint the screen.

Application Function Keys (AFKs) 18

This chapter explains how to redefine terminal keys from the application program.

18.1 What Are Application Function Keys?

Application function keys (AFKs) are keys that trigger special application-specific actions. AFKs are not restricted to the functions TDMS provides.

When the operator presses a key that the application program defines as an AFK, either or both of the following events occurs:

- An event flag is set
- A user-written asynchronous system trap (AST) routine is invoked

If you are unfamiliar with AST routines, you should read the VMS documentation on system services before continuing with this chapter.

18.2 When Do You Use Application Function Keys?

You should use AFKs when you want to associate a terminal key with an action that is unrelated to TDMS or that interrupts TDMS. For instance, you can write an AST routine that calls TSS$CANCEL and then declare an AFK that invokes that AST routine. This way, the operator can cancel a request without entering any data, even if the form defines the fields as Response Required.

18.3 Declaring Application Function Keys

You declare an AFK with the TSS$DECL_AFK call. The code to declare an AFK is as follows:

BASIC

```
Return_status = TSS$DECL_AFK ( Channel BY REF,          &
                               Key_id  BY REF,          &
                               Key_event_flag BY REF,   &
                               Key_ast_routine BY REF,  &
                               Key_ast_parameter BY REF)
```

COBOL

```
CALL "TSS$DECL_AFK"
    USING BY REFERENCE Channel,
          BY REFERENCE Key-id,
          BY REFERENCE Key-event-flag,
          BY REFERENCE Key-ast-routine,
          BY REFERENCE Key-ast-parameter,
    GIVING Return-status.
```

FORTRAN

```
 Return_status = TSS$DECL_AFK(%REF(Channel),
1                            %REF(Key_id),
2                            %REF(Key_event_flag),
3                            %REF(Key_ast_routine),
4                            %REF(Key_ast_parameter))
```

Channel is the channel number that was assigned on the TSS$OPEN call.

Key-id is a code representing the AFK. When the operator presses the key represented by the key-id parameter, the event flag will be set and the AST routine will be invoked. These parameters are optional but you must include at least one. See Table 18-1 for a list of application function keys.

Key-event-flag is the event flag that is set when the operator presses the AFK. This parameter is optional; if it is not present, TDMS does not set an event flag when the operator presses the key. However, if you do not specify an event flag, you must specify an AST routine.

Key-ast-routine is a subroutine. This parameter is optional. When the operator presses the AFK, TDMS calls this routine at AST level. You can use either the event flag or the AST routine by itself, or together.

Key-ast-parameter is a parameter to be passed to the AST routine. This parameter is optional. If the AST parameter is not present and an AST routine is, TDMS will pass an AST parameter of zero. You can pass any type of parameter you would like your AST routine to receive, including addresses.

18.3.1 Terminal Keys You Can Declare as AFKs

Table 18-1 lists the valid key codes and the keys they represent.

Table 18-1: Application Function Key Codes

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/?

18.3.2 How to Write an AST Routine

When the operator presses an AFK that has an AST routine associated with it, TDMS invokes the AST routine and passes it three parameters. You must make sure your AST routine receives the parameters correctly. The calling sequence is as follows:

```
Return-status = Key-ast-routine (Key-ast-parameter by value,
                                 channel by reference,
                                 key-id by reference)
```

18.4 Removing an AFK Key Definition

To remove a key definition declared in a TSS$DECL_AFK call, you use the TSS$UNDECL_AFK call. The code to remove a key definition is as follows.

BASIC

```
Return_status = TSS$UNDECL_AFK (Channel BY REF,    &
                                Key_id BY REF)
```

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))
```

Channel is the channel number that was assigned on the TSS$OPEN call.

Key-id is the code representing an AFK that was previously declared in a TSS$DECL_AFK call.

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.

* (asterisk)
  See Asterisk (*)
@ (at sign)
  See @file-spec command
! (exclamation point)
  See Exclamation point (!)
- (hyphen)
  See Hyphen (-)
; (semicolon)
  See Semicolon (;)

A

AFK
  See Application function keys
%ALL syntax, 3-3
  arrays
    elements of, 7-8
    horizontally-indexed, 8-7, 8-8f
    indexed, 7-15
    multiple, 7-20
    scrolled, 7-14
    two-dimensional, 8-7
  Display Only form fields, 3-6
  entire forms, 3-4
  errors in, 5-3, 5-7
  example of, 3-5
  form and larger record, 3-6
  form and smaller record, 3-7
  group record fields, 3-21
  informational messages in, 5-3
  rules, 3-3, 4-2
  when not to use, 3-19
  when to use, 3-4
  with explicit syntax, 3-19
  with INPUT TO, 1-11
  with OUTPUT TO, 1-12
Ambiguous field names, 3-14
  making unique
    using group field names, 3-15
    using record names, 3-17
    using the WITH NAME modifier, 3-17
  qualifying, 3-14
ANYMATCH case value, 6-10
Application function keys (AFKs), 18-1, 18-3t
  AST routines, 18-2, 18-4
  deassigning, 18-4
  declaring, 18-2
  event flags, 18-2

  - when to use, 18-1
- Application mode, 11-2, 11-3
- Application programs
  - canceling TDMS calls, 15-6
  - compiling, 14-9
  - data types, 16-1, 16-26t
  - debugging
    - sample, 17-3 to 17-6f
    - Trace facility, 17-1
    - using log files, 17-2
    - using two terminals, 17-6
    - VAX Symbolic Debugger, 17-6
    - with TDMS calls, 17-2
  - declaring records, 16-1
    - in BASIC programs, 16-2
    - in COBOL programs, 16-7, 16-8
    - in FORTRAN programs, 16-13, 16-14
  - linking, 14-9, 14-10
  - opening RLB files, 14-2
  - passing records to requests, 14-4, 14-5, 16-25
  - reading from the message line, 15-1
  - record definitions in, 14-5
  - sample
    - BASIC program, 14-10 to 14-12
    - COBOL program, 14-12 to 14-13
  - sequence of TDMS calls, 14-1
  - signaling errors, 14-7
  - testing return status, 14-6
  - using control values, 6-3
  - using DBMS, 16-20 to 16-22
  - using Rdb/VMS, 16-19, 16-20
- Arguments for TDMS calls, 14-1
- ARRAY clause (CDDL), 7-6
  - in BASIC programs, 16-5
  - in COBOL programs, 16-11
  - in FORTRAN programs, 16-16
  - nesting, 8-2
- Arrays, 7-1
  - %ALL syntax, 7-8, 7-14
    - horizontally-indexed, 8-7
    - two-dimensional, 8-5, 8-7
  - bounds
    - adjusting, 9-6
    - in BASIC programs, 16-6, 16-7
    - in COBOL programs, 16-11
    - in RDU, 7-7
  - control values, 9-1
    - adjusting bounds, 9-6
    - evaluating at run time, 9-2
    - example, 9-2f
    - one-dimensional, 9-7, 9-8f
    - rules for specifying, 9-4
  - declaring
    - in BASIC programs, 16-5, 16-5f, 16-7f
    - in COBOL programs, 16-11, 16-11f, 16-12f
    - in FORTRAN programs, 16-16, 16-17f
  - explicit syntax, 7-7
  - for database scrolled regions, 16-23
    - for input, 16-25
    - initializing, 16-25
    - loading, 16-25
  - form, 7-2
    - horizontally-indexed, 8-1, 8-1f
    - indexed, 7-2, 7-3f
    - mapping from multiple, 7-20
    - scrolled, 7-3, 7-4f
    - simple, 4-3f
  - horizontally-indexed, 8-1f, 8-4, 8-9
  - indexed, 7-12
  - multiple, 7-18
  - nested
    - in BASIC programs, 16-5
    - in COBOL programs, 16-11
    - in FORTRAN programs, 16-16
  - one-dimensional, 7-2f
    - as control values, 9-7, 9-8f
  - partial, 7-17, 8-10f
  - records
    - group, 7-5
    - in BASIC programs, 16-5
    - in COBOL programs, 16-11
    - in FORTRAN programs, 16-16
    - multiple, 7-18
    - one-dimensional, 7-4, 7-5f
    - structure, 4-2f, 7-4

two-dimensional, 8-2, 8-4
using ARRAY syntax, 7-6, 8-2
using OCCURS syntax, 7-6, 8-2
with database streams, 16-24
rules, 7-9
scrolled, 7-12, 7-15
displaying, 10-5, 10-6f, 10-7
horizontally-indexed, 8-1f
multiple, 7-20
structure of, 7-2f
subscripts, 7-1
See also Dependent names
ranges, 7-1
two-dimensional
%ALL syntax, 8-7
in BASIC programs, 16-7
in COBOL programs, 16-12
in FORTRAN programs, 16-18
rules, 8-4
work, 9-6
zero-based
as control values, 9-6
in BASIC programs, 16-6
in COBOL programs, 16-11
in RDU, 7-7
Arrow keys in key definitions, 11-3
AST routines
calling sequence, 18-4
for application function keys, 18-2, 18-4
for asynchronous calls, 19-2
parameters
for application function keys, 18-2
for asynchronous calls, 19-2, 19-3
writing, 18-4
Asterisk (*) character in Rdb/VMS DML, 16-20
Asynchronous calls
See TDMS programming calls
Asynchronous System Trap routines
See AST routines
Attributes
field
Display Only, 3-6
dummy fields, 10-5
Must Fill, 3-23
with CHECK modifier, 11-8
with NO CHECK modifier, 11-10
video, 1-12
in conditional instructions, 6-12
with HARDCOPY key, 15-5
with inactive forms, 5-7

B

BACK SPACE key, 10-1
Base instructions
in conditional requests, 6-2, 6-4f
in requests, 1-3
multiple OUTPUT TO instructions, 5-7
with PRKs, 11-13
BASIC programs, 14-1
CASE statement, 16-5
compiling, 14-9, 16-2
data types, 16-26t
CDDL, 16-3
declaring
record variants, 16-5
records, 16-2, 16-2f, 16-4f, 16-5f, 16-7f
subscript variables, 16-6
%INCLUDE statement, 16-2
linking, 14-9
samples, 14-10 to 14-12
using DBMS, 16-21
using Rdb/VMS, 16-19
VARIANT statement, 16-5
Bell, ringing the terminal, 15-4
Binary structures
storing, 2-9, 2-10
after modification, 2-11
BLINK FIELD instruction, 1-13
BOLD FIELD instruction, 1-13
Bounds of arrays, 7-6
adjusting, 9-6
in BASIC programs, 16-6, 16-7
in COBOL programs, 16-11
in RDU, 7-7

BUILD LIBRARY command (RDU), 12-4
  /LOG qualifier, 7-10
  Novalidate mode, 2-8
Building request libraries, 12-4 to 12-5
  errors, 12-5
  mapping messages in, 7-10
  /NOSTORE qualifier, 2-10
  Novalidate mode, 2-8
BYTE data type, 16-26t

C

Calling sequence
  asynchronous TDMS calls, 19-2
  for AST routines, 18-4
  TDMS synchronous calls, 13-2
Calls
  See TDMS programming calls
Canceling TDMS calls, 15-6, 18-1
CASE statement (VAX BASIC), 16-5
Case values
  ANYMATCH, 6-10
  case insensitivity, 6-8
  for error checking, 9-7
  in conditional requests, 6-4f
    with PRKs, 11-13
  match instructions, 6-12
  multiple control values, 6-12
  NOMATCH, 6-9
  specifying, 6-9
  structure of, 6-2
  with control arrays, 9-2
  with nested control field instructions, 6-9
CDD
  copying requests, 2-10
  DBM$RECORDS directory, 16-20
  DBM$SUBSCHEMAS directory, 16-20
  default directory
    defining CDD$DEFAULT, 2-2
    displaying, 2-3
    setting, 2-2, 2-3
  deleting requests, 2-12
  modifying requests, 2-11
  naming conventions, 2-3 to 2-4
  path names, 2-3, 16-2
    DBMS databases, 16-20
    for Rdb/VMS databases, 16-19
    in BASIC programs, 16-2
    in COBOL programs, 16-8
    in FORTRAN programs, 16-13
  RDB$RELATIONS directory, 16-19
  record definitions
    DBMS, 16-20
    in BASIC programs, 16-2
    in COBOL programs, 16-7
    in FORTRAN programs, 16-13
    Rdb/VMS, 16-19
  storing
    request library definitions, 12-1
    requests, 2-2, 2-8
CDD$DEFAULT
  defining, 2-2
    in login command file, 2-2
    in RDU, 2-3
  in BASIC programs, 16-2
  in COBOL programs, 16-8
  in FORTRAN programs, 16-13
  with CREATE REQUEST command, 2-5
  with DBMS, 16-21
  with Rdb/VMS databases, 16-19
CDDL
  ARRAY clause, 7-6
    in BASIC programs, 16-5
    in COBOL programs, 16-11
    in FORTRAN programs, 16-16
    nesting, 8-2
  array definitions, 7-1, 7-7
  COPY FROM statement, 16-23
  data types, 16-26t
    in BASIC programs, 16-3, 16-5
    in FORTRAN programs, 16-15
    TDMS, 4-6
  DESCRIPTION IS statement
    in FORTRAN programs, 16-13
  OCCURS clause, 7-6

    - in BASIC programs, 16-5
    - in COBOL programs, 16-11
    - in FORTRAN programs, 16-16
    - nesting, 8-2
  - record definitions, 1-2, 14-5
  - record names, 14-5
  - STRUCTURE statement
    - in BASIC programs, 16-3, 16-5
    - in COBOL programs, 16-9, 16-11
    - in FORTRAN programs, 16-15
  - VARIANT keyword
    - in BASIC programs, 16-5
    - in COBOL programs, 16-11
    - in FORTRAN programs, 16-16
  - VARIANTS statement
    - in BASIC programs, 16-5
    - in COBOL programs, 16-11
    - in FORTRAN programs, 16-16
- Changing
  - request library definitions, 12-3
  - requests, 2-11
- Channels
  - closing terminal, 14-6
  - opening terminal, 14-3
- CHARACTER data type, 16-26t
- CHECK modifier
  - assigning NO CHECK, 11-10
  - at run time, 11-10
  - default, 11-8
  - function of, 11-8
- CLEAR SCREEN instruction, 1-6
- Clearing the screen, 1-6, 14-6
- Closing
  - I/O channels, 14-6
  - request libraries, 14-5
- COBOL programs, 14-1
  - CDD path names in, 16-8
  - compiling, 14-9, 16-8
    - /LIST/COPY_LIST qualifiers, 16-8
  - COPY statement, 16-8
  - data types, 16-26t
  - declaring records, 16-7, 16-8, 16-9f, 16-10f, 16-11f, 16-12f
  - linking, 14-9
  - sample, 14-12 to 14-13
- Command files
  - EDT startup, 5-8
  - login, 2-1, 2-2
  - RDU
    - creating requests, 2-6
    - default file type, 2-6
    - startup, 2-3
  - RDU$EDIT, 5-9
  - TDMS$EDIT, 5-9
- Comment characters
  - in BASIC programs, 16-5
  - in DESCRIPTION instruction, 1-6
  - in FORTRAN programs, 16-13
  - TDMS, 1-6
- Common Data Definition Language
  - *See* CDDL
- Common Data Dictionary
  - *See* CDD
- COMP-1 data type, 16-26t
- COMP-2 data type, 16-26t
- Compatibility
  - of data types, 4-4, 4-7
    - for input, 4-8
  - of field length and size, 4-9
  - of field structures, 4-3
  - of input mappings, 4-8t
  - of output mappings, 4-9t
  - of scale factors, 4-5
  - of sign conditions, 4-9
  - program and request records, 14-5
- Compiling
  - BASIC programs, 16-2
  - COBOL programs, 16-8
  - FORTRAN programs, 16-13
  - TDMS programs, 14-9
- COMPLEX data type, 16-26t
- Condition codes
  - levels, 14-6
  - returned by TDMS calls, 13-2
  - signaling, 14-7
  - testing, 14-6
- Conditional instructions
  - case values
    - ANYMATCH, 6-10

NOMATCH, 6-9
specifying, 6-9
CONTROL FIELD IS, 6-4f
control values
arrays, 9-1, 9-2f
multiple, 6-12
DISPLAY FORM instructions, 6-11
evaluation of, 6-7f
run-time, 6-5
multiple, 6-8
nesting, 6-8
structure of, 6-2
USE FORM instructions, 6-11
Conditional requests, 6-1, 6-2
CONTROL FIELD IS, 6-4f
evaluation of, 6-7f
structure of, 6-3f
using PRKs to return values to control values, 11-12
when to use, 6-3
Control field arrays
illegal nesting of dependent ranges, 9-5, 9-5f
rules for specifying, 9-4
CONTROL FIELD IS instruction, 6-4f
case values
ANYMATCH, 6-10
NOMATCH, 6-9
specifying, 6-9
control values, 6-6
evaluation of, 6-7f
run-time, 6-5
match instructions, 6-12
multiple, 6-12
nesting, 6-8
Control fields
debugging, 17-1
returning a value using PRKs, 11-12
Control values
arrays, 9-1, 9-2f
dependent names, 9-1
dependent ranges, 9-1
evaluating at run time, 9-2
one-dimensional, 9-7
rules for specifying, 9-4
two-dimensional, 9-10
multiple, 6-12
record fields, 6-6
specifying, 6-6
using in the program, 6-3
workspace arrays, 9-6
workspace records, 6-6
CONTROL Z
See CTRL/Z
Controlling application flow with program request keys, 11-11
COPY LIBRARY command (RDU), 12-2
COPY REQUEST command (RDU), 2-10
COPY statement (VAX COBOL)
CDD path names in, 16-8
format of translated record definition, 16-8
general format, 16-8
Copying
from CDD to VMS file, 2-12
record definitions
into BASIC programs, 16-2
into COBOL programs, 16-7
into FORTRAN programs, 16-13
request library definitions, 12-2
requests, 2-10, 2-12
the active form, 15-4
Correcting errors, 2-7
CREATE LIBRARY command (RDU), 12-1
CREATE REQUEST command (RDU), 2-5
CDD$DEFAULT in, 2-5
errors in, 2-5
path names in, 2-5
syntax errors during, 5-2
with command file, 2-6
with database streams, 16-24
with text file, 2-6
Creating
forms, 1-2

  - records, 1-2
  - request libraries, 12-4 to 12-5
  - request library definitions, 12-1 to 12-2
  - requests, 2-5
    - command-file method, 2-6
    - from DCL level, 2-6
    - interactively, 2-5
    - text-file method, 2-6
- CTRL/Z, 2-13

D

- D_FLOATING COMPLEX data type, 16-26t
- D_FLOATING data type, 4-7t, 16-26t
- Data Manipulation Language (DML)
  - DBMS, 16-21
  - Rdb/VMS, 16-19
- Data types
  - allowable picture characters, 4-5t
  - compatibility, 4-7
    - for input, 4-8
    - of field lengths and sizes, 4-9
    - of field sign conditions, 4-9
  - conversion chart, 16-26t
  - form fields, 4-4
  - programming language support, 16-1
  - record fields, 4-6
- DATATRIEVE
  - array definitions, 7-7
  - record names, 14-5
- DATE data type, 4-7t, 16-26t
- DBG$INPUT logical name, 17-6
- DBG$OUTPUT logical name, 17-6
- DBM$RECORDS directory, 16-20
- DBM$SUBSCHEMAS directory, 16-20
- DBMS
  - DBM$RECORDS directory, 16-20
  - DBM$SUBSCHEMAS directory, 16-20
  - path names, 16-20
  - record definitions, 16-20
    - naming conventions, 16-20
    - work areas, 16-21
  - sample programs, 16-21
  - schema names, 16-20
  - subschema definitions, 16-20
  - using scrolled regions with, 16-22
- DCL commands
  - BASIC, 14-9
  - COBOL, 14-9
  - DEFINE, 2-2, 17-2
  - FORTRAN, 14-9
  - LINK, 14-9
  - TYPE, 5-9
- Deassigning application function keys, 18-4
- Deassigning key function definitions, 11-16
- Debugging TDMS programs, 17-1
  - sample, 17-3 to 17-6f
  - using log files, 17-2
  - using two terminals, 17-6
  - VAX Symbolic Debugger, 17-6
  - with TDMS calls, 17-2
- DECIMAL data type, 16-26t
- DECLARE statement (VAX BASIC), 16-2
- Declaring
  - application function keys, 18-2
  - AST routines, 18-2, 18-4
  - event flags, 18-2
  - records, 16-1
    - explicitly, 16-1
    - in BASIC programs, 16-2, 16-2f, 16-4f, 16-5f, 16-7f
    - in COBOL programs, 16-7, 16-8, 16-9f, 16-10f, 16-11f, 16-12f
    - in FORTRAN programs, 16-13, 16-14, 16-14f, 16-15f, 16-17f
  - TDMS programming calls, 14-7
- DEFAULT FIELD instruction
  - with inactive forms, 5-7
- Defaults
  - AST parameters, 18-2
  - CDD directory, 2-2
    - showing, 2-3

- dependent names, 9-2
- error message level, 5-4
- file types
  - for executable images, 14-10
  - for object files, 14-9
  - for RDU command files, 2-6
  - for request library files, 14-2
  - for the SAVE command, 5-9
- form fields
  - data types, 4-4
  - scale factor, 4-5
- I/O device, 14-3
- RDU editor, 2-7, 5-9, 12-3
- VMS directory, 14-9

DEFINE command (DCL)
- CDD$DEFAULT, 2-2
- TSS$TRACE_OUTPUT, 17-2

DEFINE KEY AS instruction, 11-1, 11-2, 11-14 to 11-17

Defining
- CDD$DEFAULT, 2-2
- forms, 1-2
- RDU
  - default editor, 2-7
  - in login command files, 2-1
  - symbol, 2-1
- RDU$EDIT, 5-9
- records, 1-2
- request libraries, 12-1
- TSS$TRACE_OUTPUT, 17-2

DELETE command (DMU), 2-12

DELETE LIBRARY command (RDU), 12-4

DELETE REQUEST command (RDU), 2-12

Deleting
- from the CDD, 2-12
- request library definitions, 12-4
- requests, 2-12

Dependent names, 9-1
- %ENTRY, 9-2
  - redefining, 9-6
- %LINE, 9-2
  - redefining, 9-6

Dependent ranges, 9-1

DESCRIPTION instruction, 1-6
- semicolon in, 1-13

DESCRIPTION IS statement (CDDL)
- in FORTRAN programs, 16-13

Dictionary Management Utility
- *See* DMU commands

DICTIONARY statement (VAX FORTRAN)
- general format, 16-13
- /LIST qualifier, 16-13

Directories
- CDD default, 2-2
- VMS default, 5-9, 12-4, 14-9

DISPLAY FORM instruction, 1-6
- errors in, 5-5, 5-7
- given names in, 2-4
- in conditional instructions, 6-11
- WITH OFFSET modifier, 5-6, 5-7

Displaying
- default CDD directory, 2-3
- forms, 1-6
  - with an offset, 5-6
- request library definitions, 12-2
- requests, 2-11
- scrolled regions, 10-5, 10-7

DML
- *See* Data Manipulation Language

DMU commands
- DELETE, 2-12
- EXTRACT, 2-12
- LIST, 4-12

Documenting requests, 1-6

DOUBLE data type, 16-26t

Down arrow key, 10-1, 11-3

E

E keys, 11-3

EDIT command (RDU), 2-7, 5-8
- default editor, 5-9

Editing
- RDU commands, 2-7
- request library definitions, 12-3
- requests, 2-11

Editor
  changing, 5-9
  default, 2-7, 5-8, 12-3
Elements of arrays, 7-1
  explicitly mapping, 7-7
Enabling the Trace facility, 17-1
END DEFINITION instruction, 1-13
Ending
  request definitions, 1-13
  request instructions, 1-13
Entering RDU, 2-1
%ENTRY lexical function, 9-1, 9-2
  redefining, 9-6
Erasing the screen, 1-6, 14-6
Error level status, 14-6
Errors
  correcting, 5-8 to 5-10
    with the SAVE command, 5-9
  form-related, 5-7
  in mappings, 5-3, 5-7
    arrays, 7-10
    explicit, 5-4
  in request library definitions, 12-2
  in TDMS programs, 14-6
  notifying the operator
    with TSS$WRITE_BRKTHRU, 15-3
    with TSS$WRITE_MSG_LINE, 15-3
  semantic, 5-3
  severity, 5-2
  signaling, 14-7
  syntax, 5-1
    how RDU reports, 5-2
    in Interactive mode, 5-2
  tracing at run time, 17-1
  while building request libraries, 12-5
Evaluating conditional instructions, 6-5
Event flags
  for asynchronous calls, 19-2
  with application function keys, 18-2
Exclamation point (!) comment character, 1-6, 1-13
Executing a request, 14-3
Execution order
  errors in, 5-6
  of conditional instructions, 6-5
EXIT command (RDU), 2-13
Exiting RDU, 2-13
Explicit mapping
  arrays, 7-7, 7-12
  errors in, 5-3, 5-4
  group record fields, 3-21
  multiple fields, 3-10
  partial arrays, 7-17
  rules, 3-3, 4-1
    for specifying fields, 3-9
  syntax, 3-3
  when to use, 3-9
  with %ALL syntax, 3-19
EXTRACT command (DMU), 2-12

F

F keys, 11-3
F_FLOATING COMPLEX data type, 16-26t
F_FLOATING data type, 4-7t, 16-26t
Fatal level status, 14-6
FDU, 1-2, 4-11
FDU commands
  LIST FORM, 4-11
Field names
  conventions, 3-3
  making unique
    using group field names, 3-15
    using record names, 3-17
    using the WITH NAME qualifier, 3-17
  qualifying, 3-14
  record arrays, 7-4
Fields
  data types, 4-4
  form field structures, 4-3
  mapping, 1-9
    %ALL syntax, 1-11
    arrays, 7-6
    multiple fields, 3-10, 3-21

restrictions, 3-23
with %ALL syntax, 1-12
naming conventions, 3-3
record structures, 4-2f, 4-3
video attributes, 1-12
FILE IS instruction
file name in, 12-1
in request library definition, 12-5
File method of creating requests, 2-6
@file-spec command (RDU), 2-6
Files
names
default for SAVE command, 5-9
request library files, 12-2, 12-4
request libraries, 12-1
TDMS shareable image, 14-9
TSS$HARDCOPY, 15-4
TSS$TRACE_OUTPUT, 17-2
Flags
See Event flags
Floating-point data types, 4-7t
Form arrays, 7-1, 7-2
horizontally-indexed, 8-1, 8-1f
indexed, 7-2, 7-3f
mapping, 7-6, 7-8
multiple, 7-20
with %ALL, 7-8
scrolled, 7-3, 7-4f
collecting data from, 10-2 to 10-5
displaying, 10-5, 10-6f, 10-7
Form Definition Utility
See FDU
Form definitions, 1-2
listing, 4-11
Form fields
data types, 4-4, 4-5t
length, 4-4
picture characters, 4-4
picture strings, 4-4
scale factors, 4-4
simple, 4-3f
structures, 4-3, 4-3f
array, 4-3
simple, 4-3
FORM IS instruction
error messages, 1-5
general format, 1-4
given names in, 2-4
in conditional requests, 6-12
path names in, 1-5, 2-4
with DISPLAY FORM, 5-5
WITH NAME modifier, 1-5, 5-6
with USE FORM, 5-5
Forms
active, 5-7
with ANYMATCH case value, 6-11
with NOMATCH case value, 6-10
CDD path names, 2-3
copying to a file, 15-4
defining, 1-2
displaying, 1-6
listing, 4-11
mapping
entire forms, 3-4
to larger record, 3-6
to smaller record, 3-7
offset on the screen, 5-6
unique names, 1-5
using in requests, 1-4
validating request references, 2-7
video attributes, 1-12
FORTRAN programs, 14-1
CDD path names in, 16-13
CDDL data types in, 16-15
comment character for CDDL descriptions, 16-13
compiling, 14-9, 16-13
contents of listing file, 16-13
data types, 16-26t
declaring
records, 16-13, 16-14, 16-14f, 16-15f, 16-17f
DICTIONARY statement
/LIST qualifier, 16-13
/G_FLOATING qualifier, 16-27
/I4 qualifier, 16-27
linking, 14-9
listing, 16-13

- MAP statement, 16-16
- two-dimensional arrays, 16-18
- UNION statement, 16-16

Full path names, 2-3
- in RDU, 2-4
- in request library definitions, 12-2

Function definition keys, 11-14 to 11-17
- examples of, 11-17

Function keys, 11-3
- application, 18-1, 18-3t
  - deassigning, 18-4
  - declaring, 18-2
  - when to use, 18-1

Functions
- *See* Lexical functions

G

G_FLOATING COMPLEX data type, 16-26t

G_FLOATING data type, 4-7t, 16-26t

GFLOAT data type, 16-26t

Given names, 2-3
- and logical names, 2-4
- in DISPLAY IS, 2-4
- in FORM IS, 2-4
- in RDU, 2-4
- in RECORD IS, 2-4
- in REQUEST IS, 2-4
- in USE FORM, 2-4

GOLD key in key definitions, 11-2 to 11-4

Group record
- with %ALL, 3-21

Group record arrays, 7-5, 7-5f
- two-dimensional, 8-3f

Group record fields, 3-15, 4-2f
- mapping, 3-21
  - restrictions, 3-23

H

H_FLOATING COMPLEX data type, 16-26t

H_FLOATING data type, 4-7t, 16-26t

HARDCOPY key, 15-4
- video attributes, 15-5
- when to use, 15-5

Header instructions
- in conditional requests, 6-2, 6-4f
- in requests, 1-3

HFLOAT data type, 16-26t

Horizontally-indexed scrolled arrays, 8-1
- %ALL syntax, 8-7
- entire array, 8-5
- partial, 8-9
- rules, 8-4
- size of, 8-1
- syntax, 8-4

Hyphen (-)
- continuation character, 2-6
- in COBOL programs, 16-8

I

I/O channels
- closing terminal, 14-6
- opening terminal, 14-3

Implicit mapping
- *See* %ALL syntax

%INCLUDE statement (VAX BASIC)
- CDD path names, 16-2
- general syntax, 16-2

Including record definitions
- in BASIC programs, 16-2
- in COBOL programs, 16-7
- in FORTRAN programs, 16-13

Indexed form arrays, 7-2, 7-3f
- %ALL syntax, 7-8, 7-15, 7-16f
- mapping partial, 7-17

Informational level messages, 5-3

Informational level status, 14-6

Input mappings, 1-9
- %ALL syntax, 1-11
- compatibility, 4-8t
  - of data types, 4-8
- from a scrolled region, 10-1
- from the message line, 15-1
- without operator input, 3-2

INPUT TO instruction, 1-9, 3-2
  %ALL syntax, 1-11
    See also %ALL syntax
  commas in, 1-9
  example of, 3-5
  in conditional requests, 6-5
  mapping arrays, 7-8, 7-15
  with PRKs, 11-11
Instances of records
  See Declaring records
Instructions
  See Request instructions
INTEGER data type, 16-26t
  with FORTRAN /I4 qualifier, 16-27
Interactive mode, 2-5
  reporting syntax errors, 5-2
Interrupting requests, 15-3
Invoking
  RDU, 2-1
  requests, 14-3
  Trace facility, 17-1

K

Key definition
  rules of precedence, 11-5
Key function definitions
  deassigning, 11-16
Keyboard keys, 11-4
KEYPAD keys in key definitions, 11-2
KEYPAD MODE IS instruction, 11-2
Keys
  See also function definition keys
  See also Program request keys
  application function keys, 18-1, 18-3t
    deassigning, 18-4
    declaring, 18-2
    when to use, 18-1
  arrow, 11-3
  BACK SPACE, 10-1
  down arrow, 10-1
  E, 11-3
  F, 11-3
  function definitions for, 11-14 to 11-17
  GOLD, 11-2 to 11-4
  HARDCOPY, 15-4
  keyboard, 11-4
  KEYPAD, 11-2
  PF, 11-2
  TAB, 10-1
  up arrow, 10-1
  viewing scrolled regions, 10-1

L

Leaving RDU, 2-13
Left arrow key, 11-3
LEFT OVERPUNCHED NUMERIC data type, 4-7t, 16-26t
LEFT SEPARATE NUMERIC data type, 4-7t, 16-26t
Length
  of form fields, 4-4
  of record fields, 4-6
Lexical functions
  %ALL
    example, 1-12
  %ALL syntax, 1-12
    See also %ALL syntax
  %ENTRY, 9-1, 9-2
  %LINE, 9-1, 9-2
Libraries
  See Request libraries
Limits of an array, 7-6
%LINE lexical function, 9-1, 9-2
  redefining, 9-6
LINK command (DCL), 14-9
Linking TDMS programs, 14-9, 14-10
LIST command (DMU), 4-12
LIST FORM command (FDU), 4-11
LIST LIBRARY command (RDU), 12-2
/LIST qualifier (VAX FORTRAN), 16-13
LIST REQUEST command (RDU), 2-11
  general format of, 2-11
  sample output, 2-12
Listing

BASIC programs with CDD records, 16-2
debugger log files, 17-2
form definitions, 4-11
FORTRAN programs with CDD records, 16-13
record definitions, 4-12
request library definitions, 12-2
requests, 2-11
/LOG qualifier
with %ALL syntax, 7-16
with BUILD LIBRARY command, 7-10
Logging errors, 17-2
Logical names
and CDD given names, 2-4
CDD$DEFAULT, 2-2, 2-3
DBG$INPUT, 17-6
DBG$OUTPUT, 17-6
RDU$EDIT, 5-9
TDMS$EDIT, 5-9
to enable Trace facility, 17-2
TSS$HARDCOPY, 15-4
TSS$TRACE_OUTPUT, 17-6
Login command files
defining CDD$DEFAULT, 2-2
defining RDU, 2-1
LONG data type, 16-26t

M

MAP statement (VAX BASIC), 16-2
MAP statement (VAX FORTRAN), 16-16
Mapping
%ALL syntax, 3-3
arrays, 7-8, 7-14, 7-15, 8-7
Display Only form fields, 3-6
entire forms, 3-4
errors in, 5-7
form and larger record, 3-6
form and smaller record, 3-7
when not to use, 3-19
when to use, 3-4
with explicit syntax, 3-19
with OUTPUT TO, 1-12
arrays, 7-6
%ALL syntax, 7-8, 7-14, 7-15, 8-7
explicit syntax, 7-12, 8-5
horizontally-indexed, 8-5
horizontally-indexed scrolled, 8-1
partial, 7-17
rules, 7-9
at run time, 14-3
compatibility
of data types, 4-4, 4-7
of field length and size, 4-9
of input, 4-8t
of output, 4-9t
of sign conditions, 4-9
errors, 5-3, 5-7
explicit syntax, 3-3
errors in, 5-4
specifying fields, 3-9
when to use, 3-9
with %ALL syntax, 3-19
group record fields, 3-21
restrictions, 3-23
multiple fields, 3-10, 3-21
restrictions, 3-13, 3-23
names
identical, 3-13
making unique, 3-13
rules, 3-3, 4-1
several form arrays, 7-20
several record arrays, 7-18
Mapping instructions, 1-9
commas in, 1-14
function of, 1-9
INPUT TO, 1-9, 3-2
OUTPUT TO, 1-11, 3-2
parentheses in, 1-14
RETURN TO, 3-2
validating, 12-5
Mapping tables
simplified input, 4-8t
simplified output, 4-9t
Match instructions, 6-12
Matching
any case value, 6-10
no case values, 6-9

Message line
  reading from, 15-1
  writing to, 15-3
MESSAGE LINE IS instruction
  in PRKs, 11-7
%MODIFIED
  RETURN TO, 3-2
Modifiers to request instructions, 1-14
MODIFY LIBRARY command
    (RDU), 12-3
MODIFY REQUEST command
    (RDU), 2-11
  correcting errors with, 5-8
  default editor, 5-9
  path names in, 2-11
Modifying
  request library definitions, 12-3
  requests, 2-11
Multiple conditional instructions, 6-8
Multiple control values, 6-12
Multiple fields
  mapping, 3-10, 3-21
    restrictions, 3-23

N

Names
  CDD path names, 2-3
  field name conventions, 3-3
  of DATATRIEVE records, 14-5
  resolving ambiguous, 3-13
  unique
    for forms, 1-5
    for records, 1-5
Naming conventions, 2-3
  fields, 3-3
  record definitions
    DBMS, 16-20
    Rdb/VMS, 16-19
  request library definitions, 12-1
  request library files, 12-2, 12-4
Nesting
  arrays, 16-5
  conditional instructions
    CONTROL FIELD IS, 6-8

NOMATCH case value, 6-9
Nonprocedural instructions, 1-4
/NOSTORE qualifier, 2-9
  in Validate mode, 2-10
Notifying the operator
  with TSS$READ_MSG_LINE,
    15-1
  with TSS$WRITE_BRKTHRU,
    15-3
  with TSS$WRITE_MSG_LINE,
    15-3
Novalidate mode
  effects of, 2-8, 2-9f
  setting, 2-8
  /STORE qualifier in, 2-9
Numeric data types, 4-7t
Numeric mode, 11-2, 11-3

O

OCCURS clause (CDDL), 7-6
  in BASIC programs, 16-5
  in COBOL programs, 16-11
  in FORTRAN programs, 16-16
  nesting, 8-2
Offsetting the form on the screen, 5-6
One-dimensional arrays, 7-2f
  as control values, 9-7
  group, 7-5f
  mapping, 7-7
  record, 7-4, 7-5f
Opening
  log files, 17-2
  request libraries, 14-2
  terminal for I/O, 14-3
Operator
  canceling TDMS calls, 18-1
  input from the message line, 15-1
  notifying
    with TSS$WRITE_BRKTHRU,
      15-3
    with TSS$WRITE_MSG_LINE,
      15-3
Optional parameters for TDMS calls,
    14-1

Order of execution of conditional instructions, 6-5
Order of TDMS programming calls, 14-1
Output mappings, 1-11
  %ALL syntax, 1-12
  compatible, 4-9t
  to the message line, 15-3
OUTPUT TO instruction, 1-11, 3-2
  %ALL syntax, 1-12
    See also %ALL syntax
  example of, 3-5
  in PRKs, 11-7
  mapping arrays, 7-8, 7-15
  %TOD, 3-2

P

PACKED DECIMAL data type, 16-26t
PACKED NUMERIC data type, 16-26t
Parameters for TDMS calls, 14-1
Partial mapping of form arrays, 7-17
Partial path names
  See Relative path names
Path names, 2-3
  for DBMS databases, 16-20
  for Rdb/VMS databases, 16-19
  full, 2-3
  given, 2-3
  in RDU, 2-4
  relative, 2-3
PF keys, 11-2
PF4 (HARDCOPY) key, 15-4
Picture characters
  and data types, 4-4, 4-5
  function of, 4-4
  resulting data type, 4-5t
Picture strings
  assigning, 4-4
  function of, 3-3
Positioning the cursor in a scrolled region, 10-3
Precedence in key definitions, 11-5
Primary TDMS calls, 14-1
PRKs
  See Program request keys
Procedure calling standard, 13-2
PROGRAM KEY IS instruction, 11-1, 11-2, 11-5 to 11-14
Program request keys, 11-5 to 11-14
  CHECK modifier, 11-8
  controlling application flow, 11-11
  creating request that uses, 11-7
  examples of, 11-11 to 11-14
  in conditional instructions, 6-12
  PROGRAM KEY IS instruction, 11-8
  returning values to control values, 11-12
  WAIT instruction with, 1-12
  when to use, 11-7
  with conditional instructions, 11-7
Programming calls
  See TDMS programming calls
Programming languages, 14-1
Programs
  See also Application programs
  canceling TDMS calls, 15-6
  compiling, 14-9
  debugging, 17-1
    sample, 17-3 to 17-6f
    using log files, 17-2
    using two terminals, 17-6
    VAX Symbolic Debugger, 17-6
  general concepts, 13-1
  linking, 14-9, 14-10
  reading from the message line, 15-1
  signaling errors, 14-7
  testing return status, 14-6
Prompts
  on the message line, 15-2
  RDU>, 2-1
  RDUDFN>, 2-6

Q

Qualifying field names, 3-14

R

Ranges of subscripts, 9-1
RDB$RELATIONS directory, 16-19
Rdb/VMS
  database path names, 16-19
  DML
    example in BASIC, 16-20
    wildcard character in, 16-20
  record definitions, 16-19
    naming conventions, 16-19
  sample program, 16-19
  using scrolled regions with, 16-22
RDU, 1-2
  correcting errors, 2-7, 5-8
  creating
    request library definitions, 12-1 to 12-2
    requests, 2-5
  default editor, 2-7
  defining symbol for, 2-1
  exiting, 2-13
  form-related errors, 5-7
  invoking, 2-1
  startup command file, 2-3
RDU commands
  BUILD LIBRARY, 12-4
  COPY LIBRARY, 12-2
  COPY REQUEST, 2-10
  CREATE LIBRARY, 12-1
  CREATE REQUEST, 2-5
  DELETE LIBRARY, 12-4
  DELETE REQUEST, 2-12
  EDIT, 2-7, 5-8
  EXIT, 2-13
  LIST LIBRARY, 12-2
  LIST REQUEST, 2-11
  MODIFY LIBRARY, 12-3
  MODIFY REQUEST, 2-11, 5-8
  SET DEFAULT, 2-3
  SET VALIDATE, 2-8
  SHOW DEFAULT, 2-3
  VALIDATE LIBRARY, 12-4
RDU editor
  changing, 5-9
  default, 5-9, 12-3
RDU error messages
  for %ALL mappings, 5-3
  syntax errors, 5-2
RDU$EDIT logical name, 2-7, 5-9
  changing, 5-9
RDUINI.COM file, 2-3
Reading from the message line, 15-1
REAL data type, 16-26t
Record arrays, 7-1, 7-4
  ARRAY syntax, 7-6
  group, 7-5, 7-5f
  mapping, 7-6
    multiple, 7-18
  OCCURS syntax, 7-6
  one-dimensional, 7-4, 7-5f
  simple, 7-4
  two-dimensional, 8-2, 8-4
    ARRAY syntax, 8-2
    OCCURS syntax, 8-2
Record definitions, 1-2
  CDD path names, 2-3
  data types, 4-6
  DBMS, 16-20
    work areas, 16-21
  declaring, 16-1
    explicitly, 16-1
    in BASIC programs, 16-2, 16-2f, 16-4f, 16-5f, 16-7f
    in COBOL programs, 16-7, 16-8, 16-9f, 16-10f, 16-11f, 16-12f
    in FORTRAN programs, 16-13, 16-14, 16-14f, 16-15f, 16-17f
  defining workspaces, 6-6
  in BASIC programs, 16-2
  in COBOL programs, 16-8
  listing, 4-12
  making fields unique, 1-5, 3-17
    using group field names, 3-15
  Rdb/VMS, 16-19
  specifying in TDMS calls, 14-4
  using in requests, 1-5
  using the WITH NAME qualifier, 3-17
  validating request references, 2-7
  with VARIANTS

    in BASIC, 16-5
Record fields
  arrays
    in BASIC programs, 16-5
    in COBOL programs, 16-11
    in FORTRAN programs, 16-16
    subscripts, 7-1
    See also Dependent names
  as control values, 6-6
  data types, 4-6, 4-7t
  length, 4-6
  making unique, 3-13
    group field names, 3-15
    record names, 3-17
    WITH NAME qualifier, 3-17
  mapping multiple fields, 3-21
  naming conventions, 3-13
  qualifying, 3-14
  scale factors, 4-6
  structures, 4-2, 4-2f
    array, 4-2
    group, 4-2
    simple, 4-2, 4-2f
RECORD IS instruction, 1-5
  given names in, 2-4
  in conditional requests, 6-12
  mapping errors in, 5-7
  path names in, 2-4
  with DBMS path names, 16-21
  WITH NAME modifier, 5-6
  with workspace records, 6-6
RECORD statement (VAX FORTRAN), 16-13
Relations
  See Rdb/VMS
Relative path names, 2-3
  in RDU, 2-4
REPLACING phrase (VAX COBOL), 16-8
Reporting errors, 14-7, 19-2
Request base, 1-3
Request Definition Utility
  See RDU
Request definitions
  terminating, 1-13
Request header, 1-3
Request instructions, 1-3
  CLEAR SCREEN, 1-6
  DEFINE KEY AS, 11-14 to 11-17
  DESCRIPTION, 1-6
  DISPLAY FORM, 1-6
  END DEFINITION, 1-13
  ending, 1-13
  FILE IS, 12-1, 12-5
  FORM IS, 1-4
  format, 1-4
  INPUT TO, 1-9, 3-2
  KEYPAD MODE IS, 11-2
  modifiers, 1-14
  order of execution, 5-6
  OUTPUT TO, 1-11, 3-2
  PROGRAM KEY IS, 11-5 to 11-14
  RECORD IS, 1-5
  REQUEST IS, 12-1
  RETURN TO, 3-2
  rules for entering, 1-13
  syntax rules, 1-13
  USE FORM, 1-7
  video field instructions, 1-12
  WAIT, 1-12
  with case values, 6-12
REQUEST IS instruction, 12-1
  given names in, 2-4
  path names in, 2-4
Request libraries, 12-1
  building, 12-4 to 12-5
  closing, 14-5
  defining, 12-1
  errors while building, 12-5
  files, 12-1
  opening, 14-2
  validating, 12-5
Request library definitions, 12-1
  building, 12-4
  copying, 12-2
  creating, 12-1 to 12-2
  deleting, 12-4
  errors in, 12-2
  listing, 12-2
  modifying, 12-3

naming conventions, 12-1
storing in the CDD, 12-1
validating, 12-3
Request library files
building, 12-4 to 12-5
closing, 14-5
default file type, 14-2
naming conventions, 12-2, 12-4
opening, 14-2
specifying, 12-5
in BUILD LIBRARY command, 12-4
in FILE IS instruction, 12-1
in RDU, 12-4
in TDMS programs, 14-2
Requests
base instructions, 1-3
binary structures, 2-9, 2-10, 2-11
building request libraries, 12-4
canceling, 15-6, 18-1
comments in, 1-6
concepts, 1-1
conditional, 6-1, 6-2
structure of, 6-3f
copying, 2-10
correcting errors, 2-7, 5-8 to 5-10
creating, 2-5
command-file method, 2-6
interactively, 2-5
text-file method, 2-6
data types, 4-7t
debugging, 17-1
deleting, 2-12
displaying a form, 1-6
ending, 1-13
executing, 14-3
format, 1-4f, 1-4
forms in, 1-4
header instructions, 1-3
interrupting, 15-3
listing, 2-11
mapping
arrays, 7-6
errors, 5-3
for input, 1-9
for output, 1-11
rules, 4-1
modifying, 2-11
naming conventions, 2-3 to 2-4
order of execution, 5-6
parts of, 1-3
passing program records to, 14-4, 14-5
records in, 1-5
request instructions, 1-3
specifying
in request libraries, 12-1
in TDMS calls, 14-4
storing, 2-2
syntax errors, 5-1
terminating, 1-14
validating, 2-7, 2-10, 12-3, 12-5
RESET FIELD instruction, 1-13
with inactive forms, 5-7
Return status
for asynchronous calls, 19-2
levels, 14-6
signaling, 14-7
synchronous calls, 13-2
testing, 14-6
Return status block (rsb) parameter, 13-2, 19-2
RETURN TO instruction, 3-2
in PRKs, 11-7
mapping arrays, 7-8, 7-15
%MODIFIED, 3-2
%TOD, 3-2
REVERSE FIELD instruction, 1-13
Right arrow key, 11-3
RIGHT OVERPUNCHED NUMERIC data type, 4-7t, 16-26t
RIGHT SEPARATE NUMERIC data type, 4-7t, 16-26t
Ringing the terminal bell, 15-4
RLB files
See also Request library files
closing, 14-5
default file type, 14-2
specifying

in RDU, 12-4
in TDMS programs, 14-2
in the request library definition, 12-5
Rsb
See Return status block (rsb) parameter
Rules
for mapping arrays, 7-9
for specifying control value arrays, 9-4
Run-time evaluation
of conditional instructions, 6-5
of control value arrays, 9-2

S

SAVE command (RDU)
correcting errors with, 5-9
default file type, 5-9
viewing saved file, 5-9
Saving the current form, 15-4
Scale factor
default, 4-5
effect on data, 4-5t
of form fields, 4-4
of record fields, 4-6
Schema
See DBMS
Screen
attributes, 1-12
clearing, 1-6, 14-6
copying, 15-4
Scrolled form arrays, 7-2, 7-3, 7-4f
%ALL syntax, 7-14
collecting data from, 10-2 to 10-5
displaying, 10-1, 10-5, 10-7
horizontally-indexed, 8-1, 8-1f, 8-5, 8-7
mapping
%ALL syntax, 7-8
explicit syntax, 7-12
multiple, 7-20
partial, 7-17
to several record arrays, 7-18
windows, 7-3
Scrolled regions with database streams, 16-22
Semantic errors, 5-3
Semicolon (;)
terminating
comment text, 1-13
DESCRIPTION instruction, 1-6
request instructions, 1-13
SET DEFAULT command (RDU), 2-3
SET VALIDATE command (RDU), 2-8
Setting event flags, 18-2
Shareable images (TSSSHR.EXE), 14-9
SHOW DEFAULT command (RDU), 2-3
Signaling errors, 14-7
with TSS$WRITE_BRKTHRU, 15-3
with TSS$WRITE_MSG_LINE, 15-3
SIGNED BYTE data type, 4-7t, 16-26t
Signed data types, 4-7t
SIGNED LONGWORD data type, 4-7t, 16-26t
SIGNED NUMERIC data type, 16-26t
SIGNED OCTAWORD data type, 16-26t
SIGNED QUADWORD data type, 4-7t, 16-26t
SIGNED WORD data type, 4-7t, 16-26t
SINGLE data type, 16-26t
Startup files
EDT, 5-8
in RDU, 2-3
Status
asynchronous calls, 13-2
for asynchronous calls, 19-2
levels, 14-6
signaling, 14-7
synchronous calls, 13-2

testing TDMS calls, 14-6
Store mode
effects of, 2-9
/STORE qualifier, 2-9
in Novalidate mode, 2-9
Storing
form images, 15-4
request library definitions, 12-1
requests, 2-8, 2-9
STRING data type, 16-26t
STRUCTURE statement (CDDL)
in BASIC programs, 16-3, 16-5
in COBOL programs, 16-9, 16-11
in FORTRAN programs, 16-15
Structures
compatibility, 4-3
form field, 4-3, 4-3f
record, 4-2, 4-2f
Subfields, 7-5
Subschema
See DBMS
Subscripts, 7-1
%ALL syntax, 8-7
arrays used as control values, 9-6
dependent ranges, 9-1
explicit syntax, 8-9
in BASIC programs, 16-6
in FORTRAN programs, 16-17
in RDU, 7-7
in two-dimensional arrays, 8-2
limits, 7-6
%LINE and %ENTRY, 9-2
assigning values to, 9-6
ranges, 7-1
See also Dependent names
specifying in array mappings, 7-7
with database streams, 16-25
Success level status, 14-6
Symbolic Debugger, 17-6
DBG$INPUT logical name, 17-6
DBG$OUTPUT logical name, 17-6
Synchronous calls
See TDMS programming calls
Syntax errors, 5-1
correcting, 2-7
how RDU reports, 5-2
in Interactive mode, 5-2
Syntax rules for request instructions, 1-13
SYS$INPUT as default terminal, 14-3

T

TAB key, 11-3
viewing scrolled regions, 10-1
Tables
See Arrays
TDMS data types, 16-26t
See Data types
TDMS programming calls
asynchronous, 13-2, 19-1
AST parameters, 19-2, 19-3
AST routines, 19-2
event flags, 19-2
parameters, 19-2 to 19-3
status block, 19-2
syntax, 19-2
when to use, 19-2
canceling, 15-6, 18-1
compiling, 14-9
debugging, 17-1
declaring, 14-7
errors in records, 5-7
format, 13-2
general concepts, 13-1
linking, 14-9, 14-10
return status, 13-2
signaling, 14-7
testing, 14-6
samples, 14-10 to 14-13
sequence, 14-1
synchronous, 13-2
TSS$CANCEL, 15-6
TSS$CLOSE, 14-6
TSS$CLOSE_RLB, 14-5
TSS$COPY_SCREEN, 15-4
TSS$DECL_AFK, 18-2
TSS$OPEN, 14-3
TSS$OPEN_RLB, 14-2
TSS$READ_MSG_LINE, 15-1

TSS$REQUEST, 14-3
TSS$SIGNAL, 14-7
TSS$TRACE_OFF, 17-2
TSS$TRACE_ON, 17-2
TSS$UNDECL_AFK, 18-4
TSS$WRITE_BRKTHRU, 15-3
TSS$WRITE_MSG_LINE, 15-3
using DBMS, 16-20 to 16-22
using Rdb/VMS, 16-19, 16-20
TDMS programs
See Application programs
TDMS shareable image, 14-9
TDMS$EDIT logical name, 5-9
Terminals
clearing the screen, 1-6, 14-6
closing, 14-6
opening, 14-3
ringing the bell, 15-4
specifying in TSS$OPEN, 14-3
using two to debug TDMS programs, 17-6
Terminating
request definitions, 1-13
request instructions, 1-13
TEXT data type, 4-7t, 16-26t
Text editor, RDU default, 2-7
Text-file method of creating requests, 2-6
%TOD
OUTPUT TO, 3-2
RETURN TO, 3-2
Trace facility, 17-1
disabling
with TSS$TRACE_OFF, 17-2
enabling, 17-1
with logical names, 17-2
with TSS$TRACE_ON, 17-2
sample, 17-3 to 17-6f
TSS$_NORMAL status code, 14-7
TSS$CANCEL programming call, 15-6
parameter, 15-7
results of, 15-7
samples, 15-7
TSS$CLOSE programming call, 14-6
parameters, 14-6
samples, 14-6
TSS$CLOSE_RLB programming call, 14-5
parameters, 14-5
samples, 14-5
TSS$COPY_SCREEN programming call, 15-4
hardcopy function, 15-4
parameters, 15-6
samples, 15-5
when to use, 15-5
TSS$DECL_AFK programming call, 18-2
parameters, 18-2
samples, 18-2
TSS$HARDCOPY logical name, 15-4, 15-6
TSS$OPEN programming call, 14-3
parameters, 14-3
samples, 14-3
TSS$OPEN_RLB programming call, 14-2
parameters, 14-2
samples, 14-2
TSS$READ_MSG_LINE programming call, 15-1
parameters, 15-2
samples, 15-2
TSS$REQUEST programming call, 14-3
parameters, 14-4 to 14-5
samples, 14-4
TSS$SIGNAL programming call, 14-7
sample, 14-7, 14-8
TSS$TRACE_OFF programming call, 17-2
TSS$TRACE_ON programming call, 17-2
samples, 17-2
TSS$TRACE_OUTPUT logical name, 17-2, 17-6
TSS$UNDECL_AFK programming call, 18-4
parameters, 18-4

samples, 18-4
TSS$WRITE_BRKTHRU programming call, 15-3
parameters, 15-4
samples, 15-3
TSS$WRITE_MSG_LINE programming call, 15-3
parameters, 15-3
samples, 15-3
TSSSHR.EXE file, 14-9
Two-dimensional arrays, 8-2
%ALL syntax, 8-7
as control values, 9-10
group, 8-2
in BASIC programs, 16-7
in COBOL programs, 16-12
in FORTRAN programs, 16-18
mapping, 8-4
partial, 8-10f
rules, 8-4

U

UNDERLINE FIELD instruction
general format, 1-12
Underlying virtual array
See Scrolled form arrays
Underscore (_)
in COBOL programs, 16-8
UNION statement (VAX FORTRAN), 16-16
Unique names
for forms, 1-5
for records, 1-5
UNSIGNED BYTE data type, 4-7t, 16-26t
Unsigned data types, 4-7t
UNSIGNED LONGWORD data type, 4-7t, 16-26t
UNSIGNED NUMERIC data type, 4-7t, 16-26t
UNSIGNED OCTAWORD data type, 16-26t
UNSIGNED QUADWORD data type, 4-7t, 16-26t
UNSIGNED WORD data type, 4-7t, 16-26t
Up arrow key, 10-1, 11-3
USE FORM instruction, 1-7
errors in, 5-5
form-related errors, 5-8
given names in, 2-4
in conditional instructions, 6-11
WITH OFFSET modifier, 5-6

V

VALIDATE LIBRARY command (RDU), 2-8, 12-4
Validate mode, 2-7
effects of, 2-9f
error checking, 5-3
/NOSTORE qualifier with, 2-10
VALIDATE REQUEST command (RDU), 2-8
Validating
mapping instructions, 12-5
request libraries, 2-8, 12-5
request library definitions, 12-3
requests, 2-7, 12-3
errors during, 5-3
VARIANT keyword (CDDL)
in BASIC programs, 16-5
in COBOL programs, 16-11
in FORTRAN programs, 16-16
VARIANTS statement (CDDL)
in BASIC programs, 16-5
in COBOL programs, 16-11
in FORTRAN programs, 16-16
VARYING TEXT data type, 4-7t
VAX data types, 16-26t
VAX Procedure Calling Standard, 13-2
VAX Symbolic Debugger, 17-6
DBG$INPUT logical name, 17-6
DBG$OUTPUT logical name, 17-6
Video attributes
in conditional instructions, 6-12
instructions to control, 1-12
with HARDCOPY key, 15-5

with inactive forms, 5-7
Viewing scrolled regions, 10-1, 10-5, 10-7
Virtual array
See Scrolled form arrays

W

WAIT instruction
general format, 1-12
with program request keys, 1-12
Warning level messages, 5-3
Wildcard character (*) in Rdb/VMS DML, 16-20
Windows, 7-3
collecting data from, 10-2 to 10-5
displaying data in, 10-5
displaying scrolled regions, 10-1
WITH NAME modifier, 1-5
making record names unique, 3-17
uniqueness of names, 5-6
WITH OFFSET modifier, 5-6, 5-7
WORD data type, 16-26t
Work areas for DBMS, 16-21
Workspace records, 6-6
arrays as control values, 9-6
Writing
AST routines, 18-4
messages
to the trace file, 17-3
to the message line, 15-3

Z

ZONED NUMERIC data type, 16-26t

VAX TDMS
Request and
Programming Manual
AA-GS14B-TE
Including: AD-GS14B-T1

Reader's Comments

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

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

____

____

____

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

____

____

____

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

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

Name ______________________________ Date ______________

Organization ______________________________

Street ______________________________

City ______________________________ State ________ Zip Code or Country ____________

-------Do Not Tear · Fold Here and Tape -------------------------------------

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 33 MAYNARD MASS

POSTAGE WILL BE PAID BY ADDRESSEE

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

-------Do Not Tear · Fold Here and Tape -------------------------------------