VAX TDMS
Pocket Guide

Order No. AA-P466B-TE

August 1986

digital equipment corporation, maynard, massachusetts

VAX TDMS
Pocket Guide

Order No. AA-P466B-TE

August 1986

This pocket guide summarizes the syntax and rules of commonly used TDMS commands, instructions, calls, and function keys.

OPERATING SYSTEM: VMS
MicroVMS

SOFTWARE VERSION: VAX TDMS V1.6

digital equipment corporation, maynard, massachusetts

Contents

Conventions

This section explains the special symbols used in this book:

[ ]	Square brackets in syntax diagrams enclose optional items from which you can choose one or none.
{ }	Braces enclose items from which you must choose one and only one alternative.
{ | | }	Bars in braces indicate that you must choose one or more of the items enclosed.
...	Horizontal ellipses indicate that you can repeat the previous item one or more times.
. . .	Vertical ellipses in an example indicate that information unrelated to the example has been omitted.
( )	In RDU syntax, matching parentheses enclose lists of receiving fields in mapping instructions and CDD passwords.
UPPERCASE	An uppercase word indicates a command or instruction keyword. Keywords are required unless otherwise indicated. Do not use keywords as variable names.
CTRL/x	This key combination indicates that you press both the CTRL (control) key and the specified key simultaneously.
GOLD-x	This key combination indicates that you press the GOLD key and the specified key consecutively.

@file-spec Command

@file-spec

COPY FORM Command

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

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

CREATE FORM Command

CREATE FORM form-path-name

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

CTRL/C Command

CTRL/C

CTRL/Y Command

CTRL/Y

CTRL/Z Command

CTRL/Z

DELETE FORM Command

DELETE FORM form-path-name

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

EDIT Command

EDIT

EXIT Command

EXIT

HELP Command

HELP [topic [subtopic]]

Command Qualifier	*Default*
/[NO]PROMPT	/PROMPT

LIST FORM Command

LIST FORM form-path-name

Command Qualifiers	*Defaults*
/OUTPUT[=file-spec]	/OUTPUT=SYS$OUTPUT
/[NO]PRINT	/NOPRINT

MODIFY FORM Command

MODIFY FORM form-path-name

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

REPLACE FORM Command

REPLACE FORM form-path-name

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

SAVE Command

SAVE file-spec

SET DEFAULT Command

SET DEFAULT CDD-directory-path-name

SET [NO]LOG Command

SET [NO]LOG [file-spec]

SET [NO]VERIFY Command

SET [NO]VERIFY

SHOW DEFAULT Command

SHOW DEFAULT

SHOW LOG Command

SHOW LOG

SHOW VERSION Command

SHOW VERSION

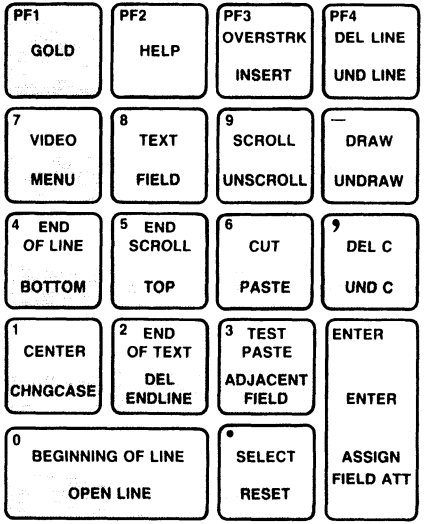
Form Phase

Form Phase Function Keys

Key	Function
TAB	Moves the cursor to the next field.
BACK SPACE (or F12)	Moves the cursor to the previous field.
LINE FEED (or F13)	Deletes the contents of the current field.
RETURN	Saves all current field values and returns you to the Phase Selection menu or saves all current values and displays the Default Field Attribute form.
MENU (GOLD-KP7)	Saves all current field values and returns you to the Phase Selection menu.
HELP (PF2 or F15)	Displays a line of help information at the bottom of the screen.

Layout Phase

Layout Phase Keypad Diagram



ZK-00095-00

Layout Phase Function Keys

Key	Function
LINE FEED (or F13)	Moves down one line.
BACK SPACE (or F12)	Moves backward one character.
HELP (PF2 or F15)	Displays help on reserved message line.
Right arrow	Moves forward one character.
Left arrow	Moves backward one character.
Up arrow	Moves up one line.
Down arrow	Moves down one line.
CTRL/U	Deletes to beginning of line.
CTRL/R	Refreshes screen.

(continued on next page)

Layout Phase Function Keys (Cont.)

Key	Function
CTRL/W	Refreshes screen.
GOLD-D	Inserts Date field.
GOLD-T	Inserts Time field.
GOLD-Q	Reverses error mode.
GOLD-S	Specifies a double-size line.
GOLD-W	Specifies a double-wide line.
GOLD-n	Repeats n times.

Picture Characters

Picture Character	Picture Type
9	Numeric (0-9)
A	Alphabetic (A-Z, a-z, space, and any alphabetic character in the DEC Multinational Character Set)
C	Alphanumeric (A-Z, a-z, 0-9, space, and any alphanumeric character in the DEC Multinational Character Set)
N	Signed numeric (0-9, +, -, period)
X	Any displayable character in the DEC Multinational Character Set

Characters from the DEC Multinational Character Set are valid for background text.

Field Constants

!	"	#	$	&	'	(	)
*	+	,	.	-	/	:	%
;	<	>	=	?	@	[	]
\	^	_	`	{	}	|	~

B (marks an embedded space)

Form Field Data Types

Picture Characters	Field Data Type
All 9s	UNSIGNED NUMERIC
All As	TEXT
All Cs	TEXT
All Ns	SIGNED NUMERIC
All Xs	TEXT
Any combination of identifiers	TEXT
All 9s or all Ns with size field validator	Named size field validator (UNSIGNED BYTE, SIGNED WORD, and so on)
Date (inserted using DATE key)	DATE
Time (inserted using TIME key)	TIME

Assign Phase

Assign Phase Function Keys

Key	Function
TAB	Moves to the next field.
BACK SPACE (or F12)	Moves to the previous field.
LINE FEED (or F13)	Deletes the contents of the field.
HELP (PF2 or F15)	Displays the help message.
CTRL/W or CTRL/R	Refreshes the screen.
RESTORE (GOLD-R)	Restores field values to the same status as when you entered the Assign phase.
RETURN or ENTER	Saves the current field values.
MENU GOLD-KP7	Saves the current field values and returns to the Phase Selection menu.

Form Field Attributes

Attribute	Function
Field Name	Assigns a name to the form field.
Default Value	Assigns a value that is displayed in the field at run time unless overridden by a request.
Help Text	Assigns a field-level help message that is displayed when the operator presses the HELP key.
Autotab	Moves the cursor automatically to the next input field when the current field is filled.
No Echo	Prevents any data from being displayed in the field.
Display Only	Prevents the field from being mapped for input.
Right Justify	Fills in a field from the right margin rather than the left margin.
Fixed Decimal	Causes data to be entered in the field first to the left of the decimal point, then to the right. Can be assigned only to an UNSIGNED NUMERIC field that has one embedded period (decimal point).

(continued on next page)

Form Field Attributes (Cont.)

Attribute	Function
Zero Fill	Determines whether the assigned fill character will be 0 or space.
Zero Suppress	Prevents a zero from being displayed on a form when a numeric record field with a null (zero) value is output to a form field.
Uppercase	Causes any alphabetic character to be displayed on the terminal and sent to the record in uppercase.
Must Fill	Requires that the field either be filled (that is, data entered in every space of the field) or be left blank.
Response Required	Requires that at least one character be entered in the field.
Clear Character	Identifies which character (usually an underscore or a blank) is displayed in any space in a field when the space does not contain data.

Scale Factor	Specifies a positive or negative integer that represents the location of a decimal point in a numeric field. (The scale factor represents a power of 10 by which the number in the form field is multiplied when data is sent to the program record.)
Indexed Field	Identifies the field as being vertically indexed (V) or horizontally indexed (H). The default is not indexed (N).
Index Count	Identifies the number of elements in a vertically or horizontally indexed field.

Form Field Validators

Validator	Function
Range	Checks operator data input against numeric or alphabetic range.
Choice	Checks operator data input against list of choices.
Size (BYTE, WORD, LONGWORD, QUADWORD)	Checks that numeric data is within certain ranges (to avoid data type conversion errors).
Check Digit	Applies a predefined algorithm to validate a numeric field.

Order Phase Function Keys

Key	Function
TAB	Moves the cursor to the next field on the access order list.
BACK SPACE (or F12)	Moves the cursor to the previous field on the access order list.
SELECT (or KP period)	Identifies the first, then subsequent, fields on the access order list.
STANDARD-ORDER (GOLD-C)	Resets all fields to be ordered from left-to-right, top-to-bottom. The access order of scrolled and indexed fields is based on the first field that is scrolled or indexed.
RESTORE (GOLD-R)	Restores the access order list to the same status as when you entered the Order phase.
MENU (GOLD-KP7)	Saves the current access order and returns you to the Phase Selection menu.
RETURN	Revises the access order list and stays in the Order phase.
GOLD-KP period	Restores the access order list to its status as when you last used the RETURN key.

@file-spec Command

@file-spec

BUILD LIBRARY Command

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

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

COPY LIBRARY Command

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

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

COPY REQUEST Command

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

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

CREATE LIBRARY Command

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

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

CREATE REQUEST Command

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

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

CTRL/C Command

CTRL/C

CTRL/Y Command

CTRL/Y

CTRL/Z Command

CTRL/Z

DELETE LIBRARY Command

DELETE LIBRARY request-library-path-name

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

DELETE REQUEST Command

DELETE REQUEST request-path-name

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

EDIT Command

EDIT

EXIT Command

EXIT

HELP Command

HELP [topic [subtopic]]

Command Qualifier	*Default*
/[NO]PROMPT	/PROMPT

LIST LIBRARY Command

LIST LIBRARY request-library-path-name

Command Qualifiers	*Defaults*
/OUTPUT[=file-spec]	/OUTPUT=SYS$OUTPUT
/[NO]PRINT	/NOPRINT

LIST REQUEST Command

LIST REQUEST request-path-name

Command Qualifiers	*Defaults*
/OUTPUT[=file-spec]	/OUTPUT=SYS$OUTPUT
/[NO]PRINT	/NOPRINT

MODIFY LIBRARY Command

MODIFY LIBRARY request-library-path-name

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

MODIFY REQUEST Command

MODIFY REQUEST request-path-name

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

REPLACE LIBRARY Command

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

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

REPLACE REQUEST Command

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

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

SAVE Command

SAVE file-spec

SET DEFAULT Command

SET DEFAULT CDD-directory-path-name

SET [NO]LOG Command

SET [NO]LOG [file-spec]

SET [NO]VALIDATE Command

SET [NO]VALIDATE

SET [NO]VERIFY Command

SET [NO]VERIFY

SHOW DEFAULT Command

SHOW DEFAULT

SHOW LOG Command

SHOW LOG

SHOW VERSION Command

SHOW VERSION

VALIDATE LIBRARY Command

VALIDATE LIBRARY request-library-path-name

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

VALIDATE REQUEST Command

VALIDATE REQUEST request-path-name

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

[NO] BLINK FIELD Instruction

[NO] BLINK FIELD { form-field[,...] | %ALL } ;

[NO] BOLD FIELD Instruction

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

[NO] CLEAR SCREEN Instruction

[NO] CLEAR SCREEN;

CONTROL FIELD IS Instruction

```
CONTROL FIELD IS record-field

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

    ...

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

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

END CONTROL FIELD;
```

[NO] DEFAULT FIELD Instruction

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

DESCRIPTION Instruction

```
                   |  /* descriptive-text */...  |
DESCRIPTION        {                            } ;
                   |  /* descriptive-text        |
                   |     descriptive-text        |
                   |     ... */                  |
```

DISPLAY FORM Instruction

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

END DEFINITION Instruction

```
END DEFINITION;
```

FILE IS Instruction

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

FORM IS Instruction

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

%INCLUDE Instruction

```
%INCLUDE "file-spec"[;]
```

INPUT TO Instruction

```
INPUT form-field TO { record-field          }
                    { (record-field[,...]) }

      [,...];

INPUT %ALL;
```

KEYPAD [MODE] IS Instruction

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

[NO] LIGHT LIST Instruction

```
[NO] LIGHT LIST number-list;
```

MESSAGE LINE IS Instruction

```
MESSAGE LINE IS { record-field  }  ;
                { quoted-string }
```

OUTPUT TO Instruction

```
OUTPUT { record-field  }  TO { form-field          }
       { quoted-string }     { (form-field[,...]) }

          [,...]    [WITH video-attribute[,...]];

OUTPUT %ALL;
```

PROGRAM KEY IS Instruction

```
PROGRAM KEY IS { GOLD    } "prk-name"  [[NO] CHECK;]
               { KEYPAD  }

{ { OUTPUT quoted-string TO form-field  [WITH video-attribute,...]; } }
{ { MESSAGE LINE IS quoted-string;                                  } }
{   RETURN quoted-string TO record-field;                           }

END PROGRAM KEY;
```

RECORD IS Instruction

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

REQUEST IS Instruction

```
{ REQUEST IS    }
{ REQUESTS ARE  }  request-path-name [WITH NAME unique-request-name],...;
```

[NO] RESET FIELD Instruction

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

RETURN TO Instruction

```
RETURN { form-field     } TO { target-record-field       }
       { quoted-string  }    { (target-record-field[,...]) }
         [,...];

RETURN %ALL;
```

[NO] REVERSE FIELD Instruction

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

[NO] RING BELL Instruction

```
[NO] RING BELL [number-of-rings];
```

SIGNAL [MODE] IS Instruction

SIGNAL [MODE] IS { BELL / REVERSE } ;

[NO] SIGNAL OPERATOR Instruction

[NO] SIGNAL OPERATOR;

[NO] UNDERLINE FIELD Instruction

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

USE FORM Instruction

USE FORM form-name [WITH OFFSET offset-value];

[NO] WAIT Instruction

[NO] WAIT;

Synchronous Calls

TSS$CANCEL Call

```
ret-status.wlc.v = TSS$CANCEL(channel.rlu.r)
```

TSS$CLOSE Call

```
ret-status.wlc.v = TSS$CLOSE(channel.rlu.r
                             [,clear-screen.rlu.r])
```

TSS$CLOSE_RLB Call

```
ret-status.wlc.v = TSS$CLOSE_RLB(library-id.rlu.r)
```

TSS$COPY_SCREEN Call

```
ret-status.wlc.v = TSS$COPY_SCREEN(channel.rlu.r
                                   ,[file-spec.rt.dx]
                                   [,append-flag.rlu.r])
```

TSS$DECL_AFK Call

```
ret-status.wlc.v = TSS$DECL_AFK(channel.rlu.r
                                ,key-id.rlu.r

                                { | ,key-efn.rlu.r      | }
                                { | ,key-astadr.szem.r  | }
                                { |  [,key-astprm.rlu.v] | }      )
```

TSS$OPEN Call

```
ret-status.wlc.v = TSS$OPEN(channel.wlu.r
                            [,device.rt.dx])
```

TSS$OPEN_RLB Call

```
ret-status.wlc.v = TSS$OPEN_RLB(rlb-file-spec.rt.dx
                                ,library-id.wlu.r)
```

TSS$READ_MSG_LINE Call

```
ret-status.wlc.v = TSS$READ_MSG_LINE(channel.rlu.r
                                     ,response-text.wt.dx
                                     ,[message-prompt.rt.dx]
                                     [,response-length.wwu.r])
```

TSS$REQUEST Call

```
ret-status.wlc.v = TSS$REQUEST(channel.rlu.r
                               ,library-id.rlu.r
                               ,request-name.rt.dx
                               [,record1.mz.r
                               [,record2.mz.r
                                  .
                                  .
                                  .
                               [,recordn.mz.r]]])
```

TSS$SIGNAL Call

```
ret-status.wlc.v = TSS$SIGNAL
```

TSS$TRACE_OFF Call

```
ret-status.wlc.v = TSS$TRACE_OFF
```

TSS$TRACE_ON Call

```
ret-status.wlc.v = TSS$TRACE_ON
```

TSS$UNDECL_AFK Call

```
ret-status.wlc.v = TSS$UNDECL_AFK(channel.rlu.r
                                  ,key-id.rlu.r)
```

TSS$WRITE_BRKTHRU Call

```
ret-status.wlc.v = TSS$WRITE_BRKTHRU(channel.rlu.r
                                     ,message-text.rt.dx
                                     [,bell-flag.rlu.r])
```

TSS$WRITE_MSG_LINE Call

```
ret-status.wlc.v = TSS$WRITE_MSG_LINE(channel.rlu.r
                                      ,message-text.rt.dx)
```

Synchronous Programming Calls in VAX BASIC

Call	Example
TSS$CANCEL	RET_STATUS = TSS$CANCEL(CHANNEL)
TSS$CLOSE	RET_STATUS = TSS$CLOSE & (CHANNEL, & CLEAR_SCREEN)
TSS$CLOSE_RLB	RET_STATUS = TSS$CLOSE_RLB & (LIBRARY_ID)
TSS$COPY_SCREEN	RET_STATUS = TSS$COPY_SCREEN & (CHANNEL, & FILE_SPEC - & APPEND_FLAG)

(continued on next page)

Synchronous Programming Calls in VAX BASIC (Cont.)

Call	Example
TSS$DECL_AFK *	RET_STATUS = TSS$DECL_AFK & (CHANNEL , & KEY_ID , & KEY_EVENT_FLAG_NUMBER , & KEY_AST_ROUTINE , & KEY_AST_PARAMETER BY VALUE)
TSS$OPEN	RET_STATUS = TSS$OPEN(CHANNEL , & DEVICE)
TSS$OPEN_RLB	RET_STATUS = TSS$OPEN_RLB & (REQUEST_LIBRARY_FILE , & LIBRARY_ID)

TSS$READ_MSG_LINE	RET_STATUS = TSS$READ_MSG_LINE & (CHANNEL, & RESPONSE_TEXT, & MESSAGE_PROMPT, & RESPONSE_LENGTH)
TSS$REQUEST	RET_STATUS = TSS$REQUEST & (CHANNEL, & LIBRARY_ID, & REQUEST_NAME, & RECORD_1, & RECORD_2, & RECORD_n)
TSS$SIGNAL	RET_STATUS = TSS$SIGNAL
TSS$TRACE_OFF	RET_STATUS = TSS$TRACE_OFF
TSS$TRACE_ON	RET_STATUS = TSS$TRACE_ON

(continued on next page)

Call	Example
TSS$UNDECL_AFK	RET_STATUS = TSS$UNDECL_AFK & (CHANNEL , & KEY_ID)
TSS$WRITE_BRKTHRU	RET_STATUS = TSS$WRITE_BRKTHRU & (CHANNEL , & MESSAGE_TEXT , & BELL_FLAG)
TSS$WRITE_MSG_LINE	RET_STATUS = TSS$WRITE_MSG_LINE & (CHANNEL , & MESSAGE_TEXT)

Note:

* Requires the following declaration at the beginning of your BASIC program:
  EXTERNAL LONG Key_ast_routine

Synchronous Programming Calls in VAX COBOL

Call	Example
TSS$CANCEL	CALL "TSS$CANCEL" USING BY REFERENCE CHANNEL, GIVING RET-STATUS.
TSS$CLOSE	CALL "TSS$CLOSE" USING BY REFERENCE CHANNEL, BY REFERENCE CLEAR-SCREEN, GIVING RET-STATUS.
TSS$CLOSE_RLB	CALL "TSS$CLOSE_RLB" USING BY REFERENCE LIBRARY-ID, GIVING RET-STATUS.

(continued on next page)

Call	Example
TSS$COPY_SCREEN	CALL "TSS$COPY_SCREEN" USING BY REFERENCE CHANNEL, BY DESCRIPTOR FILE-SPEC, BY REFERENCE APPEND-FLAG, GIVING RET-STATUS.
TSS$DECL_AFK	CALL "TSS$DECL_AFK" USING BY REFERENCE CHANNEL, BY REFERENCE KEY-ID, BY REFERENCE KEY-EVENT-FLAG-NUMBER, BY REFERENCE KEY-AST-ROUTINE, BY VALUE KEY-AST-PARAMETER, GIVING RET-STATUS.

TSS$OPEN	CALL "TSS$OPEN" USING BY REFERENCE CHANNEL, BY DESCRIPTOR DEVICE, GIVING RET-STATUS.
TSS$OPEN_RLB	CALL "TSS$OPEN_RLB" USING BY DESCRIPTOR REQUEST-LIBRARY-FILE, BY REFERENCE LIBRARY-ID, GIVING RET-STATUS.
TSS$READ_MSG_LINE	CALL "TSS$READ_MSG_LINE" USING BY REFERENCE CHANNEL, BY DESCRIPTOR RESPONSE-TEXT, BY DESCRIPTOR MESSAGE-PROMPT, BY REFERENCE RESPONSE-LENGTH, GIVING RET-STATUS.

(continued on next page)

Synchronous Programming Calls in VAX COBOL (Cont.)

Call	Example
TSS$REQUEST	CALL "TSS$REQUEST" USING BY REFERENCE CHANNEL, BY REFERENCE LIBRARY-ID, BY DESCRIPTOR REQUEST-NAME, BY REFERENCE RECORD-1, BY REFERENCE RECORD-2, BY REFERENCE RECORD-n, GIVING RET-STATUS.
TSS$SIGNAL	CALL "TSS$SIGNAL" GIVING RET-STATUS.
TSS$TRACE_OFF	CALL "TSS$TRACE_OFF" GIVING RET-STATUS.
TSS$TRACE_ON	CALL "TSS$TRACE_ON" GIVING RET-STATUS.

TSS$UNDECL_AFK	CALL "TSS$UNDECL_AFK" USING BY REFERENCE CHANNEL. BY REFERENCE KEY-ID. GIVING RET-STATUS.
TSS$WRITE_BRKTHRU	CALL "TSS$WRITE_BRKTHRU" USING BY REFERENCE CHANNEL, BY DESCRIPTOR MESSAGE-TEXT, BY REFERENCE BELL-FLAG. GIVING RET-STATUS.
TSS$WRITE_MSG_LINE	CALL "TSS$WRITE_MSG_LINE" USING BY REFERENCE CHANNEL. BY DESCRIPTOR MESSAGE-TEXT. GIVING RET-STATUS.

Call	Example
TSS$CANCEL	RET_STATUS = TSS$CANCEL(%REF(CHANNEL))
TSS$CLOSE	RET_STATUS = TSS$CLOSE(%REF(CHANNEL), 1 %REF(CLEAR_SCREEN))
TSS$CLOSE_RLB	RET_STATUS = TSS$CLOSE_RLB 1 (%REF(LIBRARY_ID))
TSS$COPY_SCREEN	RET_STATUS = TSS$COPY_SCREEN 1 (%REF(CHANNEL), 2 %DESCR(FILE_SPEC), 3 %REF(APPEND_FLAG))

TSS$DECL_AFK	RET_STATUS = TSS$DECL_AFK 1 (%REF(CHANNEL), 2 %REF(KEY_ID), 4 %REF(KEY_EVENT_FLAG_NUMBER), 5 %REF(KEY_AST_ROUTINE), 6 KEY_AST_PARAMETER)
TSS$OPEN	RET_STATUS = TSS$OPEN(%REF(CHANNEL), 1 %DESCR(DEVICE))
TSS$OPEN_RLB	RET_STATUS = TSS$OPEN_RLB 1 (%DESCR(REQUEST_LIBRARY_FILE), 2 %REF(LIBRARY_ID))
TSS$READ_MSG_LINE	RET_STATUS = TSS$READ_MSG_LINE 1 (%REF(CHANNEL), 2 %DESCR(RESPONSE_TEXT), 3 %DESCR(MESSAGE_PROMPT), 4 %REF(RESPONSE_LENGTH))

(continued on next page)

Call	Example
TSS$REQUEST	RET_STATUS = TSS$REQUEST 1 (%REF(CHANNEL), 2 %REF(LIBRARY_ID), 3 %DESCR(REQUEST_NAME), 4 %REF(RECORD_1), 5 %REF(RECORD_2), 6 %REF(RECORD_n))
TSS$SIGNAL	RET_STATUS = TSS$SIGNAL()
TSS$TRACE_OFF	RET_STATUS = TSS$TRACE_OFF()
TSS$TRACE_ON	RET_STATUS = TSS$TRACE_ON()
TSS$UNDECL_AFK	RET_STATUS = TSS$UNDECL_AFK 1 (%REF(CHANNEL), 2 %REF(KEY_ID))

TSS$WRITE_BRKTHRU	RET_STATUS = TSS$WRITE_BRKTHRU 1 (%REF(CHANNEL), 2 %DESCR(MESSAGE_TEXT), 3 %REF(BELL_FLAG))
TSS$WRITE_MSG_LINE	RET_STATUS = TSS$WRITE_MSG_LINE 1 (%REF(CHANNEL), 2 %DESCR(MESSAGE_TEXT))

TSS$CLOSE_A Call

```
ret-status.wlc.v = TSS$CLOSE_A(channel.rlu.r
                               ,[rsb.wlu.r]

                               { | ,efn.rlu.r          | }
                               { | ,astadr.szem.r      | }
                               { |   ,[astprm.rlu.v]   | }

                               [,clear-screen.rlu.r])
```

TSS$COPY_SCREEN_A Call

```
ret-status.wlc.v = TSS$COPY_SCREEN_A(channel.rlu.r
                                     ,[rsb.wlu.r]
                                     { ,efn.rlu.r          }
                                     { ,astadr.szem.r      }
                                     {   ,[astprm.rlu.v]   }
                                     ,[file-spec.rt.dx]
                                     [,append-flag.rlu.r])
```

```
ret-status.wlc.v = TSS$DECL_AFK_A(channel.rlu.r
                                  ,[rsb.wlu.r]
                                  {  ,efn.rlu.r          }
                                  {                      }
                                  {  ,astadr.szem.r      }
                                  {    ,[astprm.rlu.v]   }
                                  ,key-id.rlu.r
                                  {  ,key-efn.rlu.r      }
                                  {                      }
                                  {  ,key-astadr.szem.r  }
                                  {    [,key-astprm.rlu.v] }  )
```

TSS$OPEN_A Call

```
ret-status.wlc.v = TSS$OPEN_A(channel.wlu.r
                              ,[rsb.wlu.r]

                              { | ,efn.rlu.r                 | }
                              { |                            | }
                              { | ,astadr.szem.r            | }
                              { |   ,[astprm.rlu.v]         | }

                              [,device.rt.dx])
```

TSS$READ_MSG_LINE_A Call

```
ret-status.wlc.v = TSS$READ_MSG_LINE_A(channel.rlu.r
                                       ,[rsb.wlu.r]
                                       { ,efn.rlu.r                  }
                                       {                             }
                                       { ,astadr.szem.r              }
                                       {   ,[astprm.rlu.v]           }
                                       ,response-text.wt.dx
                                       ,[message-prompt.rt.dx]
                                       [,response-length.wwu.r])
```

TSS$REQUEST_A Call

```
ret-status.wlc.v = TSS$REQUEST_A(channel.rlu.r
                                 ,[rsb.wlu.r]
                                 { ,efn.rlu.r                  }
                                 { ,astadr.szem.r              }
                                 {   ,[astprm.rlu.v]           }
                                 ,library-id.rlu.r
                                 ,request-name.rt.dx
                                 [,record1.mz.r
                                 [,record2.mz.r
                                     .
                                     .
                                     .
                                 [,recordn.mz.r]]])
```

TSS$UNDECL_AFK_A Call

```
ret-status.wlc.v = TSS$UNDECL_AFK_A(channel.rlu.r
                                    ,[rsb.wlu.r]
                                    { ,efn.rlu.r                }
                                    { ,astadr.szem.r            }
                                    {   ,[astprm.rlu.v]         }
                                    ,key-id.rlu.r)
```

TSS$WRITE_BRKTHRU_A Call

```
ret-status.wlc.v = TSS$WRITE_BRKTHRU_A(channel.rlu.r
                                      ,[rsb.wlu.r]

                                      { |,efn.rlu.r                 | }
                                      { |                           | }
                                      { |,astadr.szem.r             | }
                                      { |  ,[astprm.rlu.v]           | }

                                      ,message-text.rt.dx
                                      [,bell-flag.rlu.r])
```

TSS$WRITE_MSG_LINE_A Call

```
ret-status.wlc.v = TSS$WRITE_MSG_LINE_A(channel.rlu.r
                                        ,[rsb.wlu.r]
                                        { | ,efn.rlu.r          | }
                                        { | ,astadr.szem.r      | }
                                        { | ,[astprm.rlu.v]     | }
                                        ,message-text.rt.dx)
```

Asynchronous Programming Calls in VAX BASIC

Call	Example
TSS$CLOSE_A +	RET_STATUS = TSS$CLOSE_A & (CHANNEL , & RETURN_STATUS_BLOCK , & EVENT_FLAG_NUMBER , & AST_ROUTINE , & AST_PARAMETER BY VALUE , & CLEAR_SCREEN)
TSS$COPY_SCREEN_A +	RET_STATUS = TSS$COPY_SCREEN_A & (CHANNEL , & RETURN_STATUS_BLOCK , & EVENT_FLAG_NUMBER , & AST_ROUTINE , & AST_PARAMETER BY VALUE , & FILE_SPEC , & APPEND_FLAG)

(continued on next page)

Call	Example
TSS$DECL_AFK_A * +	RET_STATUS = TSS$DECL_AFK_A & (CHANNEL, & RETURN_STATUS_BLOCK, & EVENT_FLAG_NUMBER, & AST_ROUTINE, & AST_PARAMETER BY VALUE, & KEY_ID, & KEY_EVENT_FLAG_NUMBER, & KEY_AST_ROUTINE, & KEY_AST_PARAMETER BY VALUE)

TSS$OPEN_A +	RET_STATUS = TSS$OPEN_A & (CHANNEL , & RETURN_STATUS_BLOCK , & EVENT_FLAG_NUMBER , & AST_ROUTINE , & AST_PARAMETER BY VALUE , & DEVICE)
TSS$READ_MSG_LINE_A +	RET_STATUS = TSS$READ_MSG_LINE_A & (CHANNEL , & RETURN_STATUS_BLOCK , & EVENT_FLAG_NUMBER , & AST_ROUTINE , & AST_PARAMETER BY VALUE , & RESPONSE_TEXT , & MESSAGE_PROMPT , & RESPONSE_LENGTH)

(continued on next page)

Call	Example
TSS$REQUEST_A +	RET_STATUS = TSS$REQUEST_A & (CHANNEL, & RETURN_STATUS_BLOCK, & EVENT_FLAG_NUMBER, & AST_ROUTINE, & AST_PARAMETER BY VALUE, & LIBRARY_ID, & REQUEST_NAME, & RECORD_1, & RECORD_2, & RECORD_n)

TSS$UNDECL_AFK_A +	RET_STATUS = TSS$UNDECL_AFK_A & (CHANNEL , & RETURN_STATUS_BLOCK , & EVENT_FLAG_NUMBER , & AST_ROUTINE , & AST_PARAMETER BY VALUE , & KEY_ID)
TSS$WRITE_BRKTHRU_A +	RET_STATUS = TSS$WRITE_BRKTHRU_A & (CHANNEL , & RETURN_STATUS_BLOCK , & EVENT_FLAG_NUMBER , & AST_ROUTINE , & AST_PARAMETER BY VALUE , & MESSAGE_TEXT , & BELL_FLAG)

(continued on next page)

Call	Example
TSS$WRITE_MSG_LINE_A +	RET_STATUS = TSS$WRITE_MSG_LINE_A & (CHANNEL , & RETURN_STATUS_BLOCK , & EVENT_FLAG_NUMBER , & AST_ROUTINE , & AST_PARAMETER BY VALUE , & MESSAGE_TEXT)

Notes:

* Requires the following declaration at the beginning of your BASIC program:

  EXTERNAL LONG Key_ast_routine

+ Requires the following declaration at the beginning of your BASIC program:

  EXTERNAL LONG Ast_routine

Asynchronous Programming Calls in VAX COBOL

Call	Example
TSS$CLOSE_A	CALL "TSS$CLOSE_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY REFERENCE CLEAR-SCREEN, GIVING RET-STATUS.

(continued on next page)

Call	Example
TSS$COPY_SCREEN_A	CALL "TSS$COPY_SCREEN_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY DESCRIPTOR FILE-SPEC, BY REFERENCE APPEND-FLAG, GIVING RET-STATUS.

TSS$DECL_AFK_A	CALL "TSS$DECL_AFK_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY REFERENCE KEY-ID, BY REFERENCE KEY-EVENT-FLAG-NUMBER, BY REFERENCE KEY-AST-ROUTINE, BY VALUE KEY-AST-PARAMETER, GIVING RET-STATUS.
TSS$OPEN_A	CALL "TSS$OPEN_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY DESCRIPTOR DEVICE, GIVING RET-STATUS.

(continued on next page)

Call	Example
TSS$READ_MSG_LINE_A	CALL "TSS$READ_MSG_LINE_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY DESCRIPTOR RESPONSE-TEXT, BY DESCRIPTOR MESSAGE-PROMPT, BY REFERENCE RESPONSE-LENGTH, GIVING RET-STATUS.

TSS$REQUEST_A	CALL "TSS$REQUEST_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY REFERENCE LIBRARY-ID, BY DESCRIPTOR REQUEST-NAME, BY REFERENCE RECORD-1, BY REFERENCE RECORD-2, BY REFERENCE RECORD-n, GIVING RET-STATUS.

(continued on next page)

Call	Example
TSS$UNDECL_AFK_A	CALL "TSS$UNDECL_AFK_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY REFERENCE KEY-ID, GIVING RET-STATUS.

TSS$WRITE_BRKTHRU_A	CALL "TSS$WRITE_BRKTHRU_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY DESCRIPTOR MESSAGE-TEXT, BY REFERENCE BELL-FLAG, GIVING RET-STATUS.
TSS$WRITE_MSG_LINE_A	CALL "TSS$WRITE_MSG_LINE_A" USING BY REFERENCE CHANNEL, BY REFERENCE RETURN-STATUS-BLOCK, BY REFERENCE EVENT-FLAG-NUMBER, BY REFERENCE AST-ROUTINE, BY VALUE AST-PARAMETER, BY DESCRIPTOR MESSAGE-TEXT, GIVING RET-STATUS.

Call	Example
TSS$CLOSE_A	RET_STATUS = TSS$CLOSE_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %REF(CLEAR_SCREEN))

TSS$COPY_SCREEN_A	RET_STATUS = TSS$COPY_SCREEN_A 1 ('%REF(CHANNEL'), 2 %REF(RETURN_STATUS_BLOCK'), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE'), 5 AST_PARAMETER, 6 %DESCR(FILE_SPEC), 7 %REF(APPEND_FLAG))
TSS$DECL_AFK_A	RET_STATUS = TSS$DECL_AFK_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %REF(KEY_ID), 7 %REF(KEY_EVENT_FLAG_NUMBER), 8 %REF(KEY_AST_ROUTINE), 9 KEY_AST_PARAMETER)

(continued on next page)

Call	Example
TSS$OPEN_A	RET_STATUS = TSS$OPEN_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %DESCR(DEVICE))
TSS$READ_MSG_LINE_A	RET_STATUS = TSS$READ_MSG_LINE_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %DESCR(RESPONSE_TEXT), 7 %DESCR(MESSAGE_PROMPT), 8 %REF(RESPONSE_LENGTH))

TSS$REQUEST_A	RET_STATUS = TSS$REQUEST_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %REF(LIBRARY_ID), 7 %DESCR(REQUEST_NAME), 8 %REF(RECORD_1), 9 %REF(RECORD_2), a %REF(RECORD_n))
TSS$UNDECL_AFK_A	RET_STATUS = TSS$UNDECL_AFK_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %REF(KEY_ID))

(continued on next page)

Call	Example
TSS$WRITE_BRKTHRU_A	RET_STATUS = TSS$WRITE_BRKTHRU_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %DESCR(MESSAGE_TEXT), 7 %REF(BELL_FLAG))
TSS$WRITE_MSG_LINE_A	RET_STATUS = TSS$WRITE_MSG_LINE_A 1 (%REF(CHANNEL), 2 %REF(RETURN_STATUS_BLOCK), 3 %REF(EVENT_FLAG_NUMBER), 4 %REF(AST_ROUTINE), 5 AST_PARAMETER, 6 %DESCR(MESSAGE_TEXT))

Data Type Conversion Chart

VAX Data Type	CDDL Data Type	TDMS Data Type	BASIC Data Type	COBOL Data Type	FORTRAN Data Type
SIGNED BYTE	SIGNED BYTE	SIGNED BYTE	BYTE	Unsupported	BYTE
SIGNED WORD	SIGNED WORD	SIGNED WORD	WORD	PIC S9(1-4) COMP	INTEGER*2
SIGNED LONGWORD	SIGNED LONGWORD	SIGNED LONGWORD	LONG	PIC S9(5-9) COMP	INTEGER*4 or INTEGER
SIGNED QUADWORD	SIGNED QUADWORD	SIGNED QUADWORD	Unsupported	PIC S9(10-18) COMP	Unsupported
SIGNED OCTAWORD	SIGNED OCTAWORD	Unsupported	Unsupported	Unsupported	Unsupported

(continued on next page)

Data Type Conversion Chart (Cont.)

VAX Data Type	CDDL Data Type	TDMS Data Type	BASIC Data Type	COBOL Data Type	FORTRAN Data Type
UNSIGNED BYTE	UNSIGNED BYTE	UNSIGNED BYTE	Unsupported	Unsupported	Unsupported
UNSIGNED WORD	UNSIGNED WORD	UNSIGNED WORD	Unsupported	PIC 9(4) COMP	Unsupported
UNSIGNED LONGWORD	UNSIGNED LONGWORD	UNSIGNED LONGWORD	Unsupported	PIC 9(9) COMP	Unsupported
UNSIGNED QUADWORD	UNSIGNED QUADWORD	Unsupported	Unsupported	PIC 9(18) COMP	Unsupported
UNSIGNED OCTAWORD	UNSIGNED OCTAWORD	Unsupported	Unsupported	Unsupported	Unsupported
F_FLOATING	F_FLOATING	F_FLOATING	SINGLE	COMP-1	REAL or REAL*4
D_FLOATING	D_FLOATING	D_FLOATING	DOUBLE	COMP-2	REAL*8

G_FLOATING	G_FLOATING	G_FLOATING	GFLOAT	Unsupported	REAL*8
H_FLOATING	H_FLOATING	H_FLOATING	HFLOAT	Unsupported	REAL*16
F_FLOATING COMPLEX	F_FLOATING COMPLEX	Unsupported	Unsupported	Unsupported	COMPLEX or COMPLEX*8
D_FLOATING COMPLEX	D_FLOATING COMPLEX	Unsupported	Unsupported	Unsupported	COMPLEX*16
G_FLOATING COMPLEX	G_FLOATING COMPLEX	Unsupported	Unsupported	Unsupported	COMPLEX*16
H_FLOATING COMPLEX	H_FLOATING COMPLEX	Unsupported	Unsupported	Unsupported	Unsupported
CHARACTER	TEXT	TEXT	STRING	PIC X(n)	CHARACTER*n

(continued on next page)

Data Type Conversion Chart (Cont.)

VAX Data Type	CDDL Data Type	TDMS Data Type	BASIC Data Type	COBOL Data Type	FORTRAN Data Type
UNSIGNED NUMERIC	UNSIGNED NUMERIC	UNSIGNED NUMERIC	Unsupported	PIC 9(m)V9(n)	Unsupported
LEFT SEPARATE NUMERIC	LEFT SEPARATE NUMERIC	LEFT SEPARATE NUMERIC	Unsupported	PIC S9(m)V9(n) LEADING SEPARATE	Unsupported
RIGHT SEPARATE NUMERIC	RIGHT SEPARATE NUMERIC	RIGHT SEPARATE NUMERIC	Unsupported	PIC S9(m)V9(n) TRAILING SEPARATE	Unsupported
LEFT OVERPUNCHED NUMERIC	LEFT OVERPUNCHED NUMERIC	LEFT OVERPUNCHED NUMERIC	Unsupported	PIC S9(m)V9(n) LEADING	Unsupported
RIGHT OVERPUNCHED NUMERIC	RIGHT OVERPUNCHED NUMERIC	RIGHT OVERPUNCHED NUMERIC	Unsupported	PIC S9(m)V9(n) TRAILING	Unsupported

ZONED NUMERIC	SIGNED NUMERIC	ZONED NUMERIC	Unsupported	Unsupported	Unsupported
PACKED DECIMAL	PACKED NUMERIC	PACKED DECIMAL	DECIMAL	PIC S9(m)V9(n) COMP-3	Unsupported
DATE	DATE	DATE	Unsupported	PIC S9(11)V9(7)	Unsupported

Notes to Data Type Conversion Chart:

- COBOL has no exact equivalent for the unsigned integer data types UNSIGNED WORD and UNSIGNED LONGWORD. The COBOL compiler issues a warning diagnostic and treats the item as an unsigned COMP data type.
- The FORTRAN data type INTEGER corresponds to SIGNED LONGWORD if you compile the program with the default qualifier /I4. If you compile the program with the /NOI4 qualifier, INTEGER corresponds to the TDMS data type SIGNED WORD.
- The FORTRAN data type REAL_*8 corresponds to G_FLOATING if you compile the program with the /G_FLOATING qualifier; otherwise, REAL_*8 corresponds to the TDMS data type D_FLOATING.
- The FORTRAN data type COMPLEX_*16 corresponds to G_FLOATING COMPLEX if you compile the program with the /G_FLOATING qualifier; otherwise, COMPLEX_*16 corresponds to the VAX data type D_FLOATING COMPLEX.

TDMS Input Mappings (Form Fields to Record Fields)

Y = Input mapping always permitted. N = Input mapping never permitted. 1-15 = Mapping permitted, subject to conditions. (See key following the table for explanation.)

Form Field Data Type	Record Field Data Types																				
	BU	WU	LU	B	W	L	Q	F	D	G	H	T	VT	GT	NU	NL	NLO	NR	NRO	NZ	P	ADT
UNSIGNED BYTE	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	2	2	1	2	2	2	2	2	2	2	N
UNSIGNED WORD	N	Y	Y	N	N	Y	Y	Y	Y	Y	Y	3	3	1	3	3	3	3	3	3	3	N
UNSIGNED LONGWORD	N	N	Y	N	N	N	Y	Y	Y	Y	Y	4	4	1	4	4	4	4	4	4	4	N
SIGNED BYTE	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	5	5	1	N	5	2	5	2	2	2	N
SIGNED WORD	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	6	6	1	N	6	3	6	3	3	3	N
SIGNED LONGWORD	N	N	N	N	N	Y	Y	Y	Y	Y	Y	7	7	1	N	7	4	7	4	4	4	N
SIGNED QUADWORD	N	N	N	N	N	N	Y	Y	Y	Y	Y	9	9	1	N	9	8	9	8	8	8	N
TEXT	N	N	N	N	N	N	N	N	N	N	N	1	1	1	N	N	N	N	N	N	N	N
UNSIGNED NUMERIC	10	11	12	10	11	12	13	14	14	Y	Y	1	1	15	1	1	1	1	1	1	1	N
SIGNED NUMERIC	N	N	N	10	11	12	13	14	14	Y	Y	1	1	15	N	1	1	1	1	1	1	N
DATE (Format 1)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	N	N	N	N	N	N	N	Y
DATE (Format 2)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	N	N	N	N	N	N	N	Y
DATE (Format 3)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	1	N	N	N	N	N	1	Y
DATE (Format 4)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	1	N	N	N	N	N	1	Y
DATE (Format 5)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	N	N	N	N	N	N	N	Y
TIME (Format 1)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	1	N	N	N	N	N	1	Y
TIME (Format 2)	N	N	N	N	N	N	N	N	N	N	N	1	1	1	N	N	N	N	N	N	N	Y

Record Field Data Types:

BU	– UNSIGNED BYTE	Q	– SIGNED QUADWORD	VT	– VARYING TEXT	NR	– RIGHT SEPARATE NUMERIC
WU	– UNSIGNED WORD	F	– F__FLOATING	GT	– GROUP TEXT	NRO	– RIGHT OVERPUNCHED NUMERIC
LU	– UNSIGNED LONGWORD	D	– D__FLOATING		(non-elementary item)	NZ	– SIGNED NUMERIC (zoned sign)
B	– SIGNED BYTE	G	– G__FLOATING	NU	– UNSIGNED NUMERIC	P	– PACKED DECIMAL
W	– SIGNED WORD	H	– H__FLOATING	NL	– LEFT SEPARATE NUMERIC	ADT	– DATE
L	– SIGNED LONGWORD	T	– TEXT	NLO	– LEFT OVERPUNCHED NUMERIC		

Key for Input Mappings:

1 – Permitted only if length of record field is greater than or equal to length of form field.
2 – Permitted only if record field has length of at least 3 bytes.
3 – Permitted only if record field has length of at least 5 bytes.
4 – Permitted only if record field has length of at least 10 bytes.
5 – Permitted only if record field has length of at least 4 bytes.
6 – Permitted only if record field has length of at least 6 bytes.
7 – Permitted only if record field has length of at least 11 bytes.
8 – Permitted only if record field has length of at least 19 bytes.
9 – Permitted only if record field has length of at least 20 bytes.
10 – Permitted only if form field has no more than 2 characters.
11 – Permitted only if form field has no more than 4 characters.
12 – Permitted only if form field has no more than 9 characters.
13 – Permitted only if form field has no more than 18 characters.
14 – Permitted only if form field has no more than 38 characters.
15 – Permitted only if length of record field is greater than or equal to length of form field; RDU also issues a warning message possible conversion error on group move.

TDMS Output Mappings (Record Fields to Form Fields)

Y = Output mapping always permitted. N = Output mapping never permitted. 1-29 = Mapping permitted, subject to conditions. (See key following the table for explanation.)

Form Field Data Type	Record Field Data Types																				
	BU	WU	LU	B	W	L	Q	F	D	G	H	T	VT	GT	NU	NL	NLO	NR	NRO	NZ	P	ADT
UNSIGNED BYTE	Y	15	15	16	1	1	1	N	N	N	N	N	N	14	18	25	25	25	25	25	25	N
UNSIGNED WORD	Y	Y	15	16	16	1	1	N	N	N	N	N	N	14	19	26	26	26	26	26	26	N
UNSIGNED LONGWORD	Y	Y	Y	16	16	16	1	N	N	N	N	N	N	14	20	27	27	27	27	27	27	N
SIGNED BYTE	15	15	15	Y	15	15	15	N	N	N	N	N	N	14	18	18	18	18	18	18	18	N
SIGNED WORD	Y	15	15	Y	Y	15	15	N	N	N	N	N	N	14	19	19	19	19	19	19	19	N
SIGNED LONGWORD	Y	Y	15	Y	Y	Y	15	N	N	N	N	N	N	14	20	20	20	20	20	20	20	N
SIGNED QUADWORD	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	14	21	21	21	21	21	21	21	N
TEXT	2	3	4	5	6	7	12	17	17	17	17	13	13	13	13	13	24	13	24	24	24	N
UNSIGNED NUMERIC	2	3	4	8	9	10	11	N	N	N	N	29	29	14	13	23	28	23	28	28	28	N
SIGNED NUMERIC	2	3	4	5	6	7	12	N	N	N	N	29	29	14	13	13	24	13	24	24	24	N
DATE (Format 1)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	N	N	N	N	N	N	N	Y
DATE (Format 2)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	N	N	N	N	N	N	N	Y
DATE (Format 3)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	13	N	N	N	N	N	13	Y
DATE (Format 4)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	13	N	N	N	N	N	13	Y
DATE (Format 5)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	N	N	N	N	N	N	N	Y
TIME (Format 1)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	13	N	N	N	N	N	13	Y
TIME (Format 2)	N	N	N	N	N	N	N	N	N	N	N	13	13	14	N	N	N	N	N	N	N	Y

Record Field Data Types:

BU	– UNSIGNED BYTE	Q	– SIGNED QUADWORD	VT	– VARYING TEXT	NR	– RIGHT SEPARATE NUMERIC
WU	– UNSIGNED WORD	F	– F__FLOATING	GT	– GROUP TEXT	NRO	– RIGHT OVERPUNCHED NUMERIC
LU	– UNSIGNED LONGWORD	D	– D__FLOATING		(non-elementary item)	NZ	– SIGNED NUMERIC (zoned sign)
B	– SIGNED BYTE	G	– G__FLOATING	NU	– UNSIGNED NUMERIC	P	– PACKED DECIMAL
W	– SIGNED WORD	H	– H__FLOATING	NL	– LEFT SEPARATE NUMERIC	ADT	– DATE
L	– SIGNED LONGWORD	T	– TEXT	NLO	– LEFT OVERPUNCHED NUMERIC		

Key for Output Mappings:

1 – Possible overflow; negative value in record field produces run-time error. RDU issues warning message.
2 – RDU issues a warning message if form field is less than 3 characters (possible overflow).
3 – RDU issues a warning message if form field is less than 5 characters (possible overflow).
4 – RDU issues a warning message if form field is less than 10 characters (possible overflow).
5 – RDU issues a warning message if form field is less than 4 characters (possible overflow).
6 – RDU issues a warning message if form field is less than 6 characters (possible overflow).
7 – RDU issues a warning message if form field is less than 11 characters (possible overflow).
8 – Possible overflow if form field is less than 3 characters; negative value in record produces run-time error. RDU issues a warning message.
9 – Possible overflow if form field is less than 5 characters; negative value in record produces run-time error. RDU issues a warning message.
10 – Possible overflow if form field is less than 10 characters; negative value in record produces run-time error. RDU issues a warning message.
11 – Possible overflow if form field is less than 19 characters; negative value in record produces run-time error. RDU issues a warning message.
12 – RDU issues a warning message if form field is less than 20 characters (possible overflow).
13 – RDU issues a warning message if length of record field is greater than length of form field (possible overflow).
14 – RDU issues a warning message if length of record field is greater than length of form field (possible overflow, possible error on group move).
15 – RDU issues a warning message (possible overflow).
16 – RDU issues a warning message; negative value in record field produces run-time error.
17 – Permitted only if form field is at least 8 characters in length; display format is in scientific notation.
18 – RDU issues a warning message if record field is more than 2 bytes.
19 – RDU issues a warning message if record field is more than 4 bytes.
20 – RDU issues a warning message if record field is more than 9 bytes.
21 – RDU issues a warning message if record field is more than 18 bytes.
22 – RDU issues a warning message if record field is more than 38 bytes.
23 – RDU issues a warning message if record field is more than 1 character longer than form field. Negative value in record produces run-time error.
24 – RDU issues a warning message if form field is not at least 1 character longer than record field.
25 – RDU issues a warning message if record field is more than 2 bytes. Negative value in record field produces run-time error.
26 – RDU issues a warning message if record field is more than 4 bytes. Negative value in record field produces run-time error.
27 – RDU issues a warning message if record field is more than 10 bytes. Negative value in record field produces run-time error.
28 – RDU issues a warning message if length of record field is greater than length of form field (possible overflow). Negative value in record field produces run-time error.
29 – RDU issues warning message; data output to form may be inconsistent with form field picture string.