HP 3000 Computer Systems

HP TRANSACT
Quick Reference Guide

HP Part No. 32247-90020
Printed in U.S.A.

Second Edition
E0494

Contents

1

Transact Verbs

CALL

CALL *file-name*[([*password*,][*mode*])][,*option-list*];

OPTIONS
DATA=*item-name*[(*subscript*)]
SIZE=*number*
SWAP
INFORM
REPORT
STATUS

Example

```
CALL INVMGT ("PASS",7),DATA=ORDER,SIZE=1000,STATUS;
```

CLOSE

CLOSE *source*[,*option-list*];

SOURCE

file-name	- MPE or KSAM file
$FORMLIST	- VPLUS spool file
$PRINT	- TRANLIST print file
$VPLS	- active formfile
set-name(base-name)	- IMAGE dataset
@[(*base-name*)]	- IMAGE database

OPTIONS
ERROR=*label*([*item-name*])
NOMSG
STATUS

Example

```
CLOSE KDATA;
```

DATA

DATA[(*modifier*)][*item-name*][("*prompt-string*")][,*option-list*][:*item-name*...]...;

MODIFIERS

```
none   - Places value in data register
ITEM   - Prompts for item name and places value in data register
KEY    - Places value in argument register
MATCH  - Places value in data and match registers
PATH   - Places value in data and argument registers
SET    - Places value in data register if entered
UPDATE - Places value in data and update registers
```

OPTIONS

BLANKS
CHECK=*set-name*
CHECKNOT=*set-name*
NOECHO
NULL
RIGHT
STATUS

Example

```
DATA CUSTNUM ("Please enter customer number"),BLANKS,CHECK=CUST-M;
```

DEFINE

DEFINE(*modifier*) *definition-list*;

MODIFIERS

ENTRY - Defines a program control label
INTRINSIC - Defines MPE intrinsic to be called
ITEM - Defines one or more items, a synonym, a marker item

DEFINITION WITH DEFINE(ENTRY)

label[:*label*]

DEFINITION WITH DEFINE(INTRINSIC)

intrinsic-name[:*intrinsic-name*]

DEFINITION WITH DEFINE(ITEM)

item-name[*count*]
[*type*(*size*[,*decimal-length*[,*storage-length*]])]
[=*parent-name*[(*position*)]]
[,ALIAS=(*alias-reference*)]
[,COMPUTE=*arithmetic-expression*]
[,EDIT="*edit-mask*"]
[,ENTRY="*entry-text*"]
[,HEAD="*heading-text*"]
[,INIT=[*value*|(BINARY(*value*))|(HEX(*value*))|(OCTAL(*value*))]
[,OPT]
[:*item-name*...]

Example

```
DEFINE(ITEM) TOTAL I(4,0,2),COMPUTE=[(QTY)*(PRICE)];
```

DELETE

```
DELETE[(modifier)]source[,option-list];
```

MODIFIERS

```
none    - Deletes from master set
CHAIN   - Deletes from detail set or KSAM file
          based on key
CURRENT - Deletes last entry accessed from KSAM file
          or data set
DIRECT  - Deletes entry at specified record number in KSAM
          file or data set
PRIMARY - Deletes from master set at the primary location
RCHAIN  - Same as CHAIN in reverse order for a data set;
          same as CHAIN for KSAM file
RSERIAL - Same as SERIAL in reverse order for data set;
          same as SERIAL for KSAM file
SERIAL  - Deletes entries in serial mode
```

SOURCE

file-name
set-name[(*base-name*)]

OPTIONS

ERROR=*label*[([*item-name*])]
LIST=(*range-list*)
LOCK
NOCOUNT
NOMATCH
NOMSG
PERFORM=*label*
RECNO=*item-name*[(*subscript*)]
SINGLE
SOPT
STATUS

Example

```
DELETE(CHAIN) ORDER-D,NOMATCH,PERFORM=PRINT-PARA;
```

DISPLAY

DISPLAY [([*TABLE*][,FILE=*mpe-file*])][*display-list*] ... ;

DISPLAY LIST
display-field[,*format-parm*]...
[:*display-field*[,*format-parm*]...]

DISPLAY FIELDS
item-name[(*subscript*)]
"*string*"
$CPU
$DATELINE
$PAGE
$TIME
$TODAY

FORMAT PARAMETER
CCTL=*number*
CENTER
COL=*number*
EDIT="*edit-string*"
HEAD="*character-string*"
JOIN[=*number*]
LEFT
LINE[=*number*]
LNG=*number*
NEED=*number*
NOCRLF
NOHEAD
NOSIGN
PAGE[=*number*]
RIGHT
ROW=*number*
SPACE[=*number*]
TITLE
TRUNCATE
ZERO[E]S

Example

```
DISPLAY CUSTNUM,LINE=1,NOHEAD,COL=10;
```

END

END [*modifier*];

MODIFIERS

none	- Ends the command sequence
(LEVEL)	- Ends the current level
system-name	- Ends the executing program file
(SEQUENCE)	- Returns to command level

Example

```
END PROG1;
```

EXIT

EXIT;

Example

```
EXIT;
```

FILE

FILE(*modifier*) *file-name*[,*option-list*];

MODIFIERS

```
CLOSE   -  Closes the file
CONTROL -  Performs an FCONTROL operation
OPEN    -  Opens the file
READ    -  Reads record from the file
SORT    -  Sorts the file
UPDATE  -  Replaces current record
WRITE   -  Writes record to file
```

OPTIONS WITH CONTROL

CODE=*number*

PARM=*item-name*[(*subscript*)]

OPTION WITH OPEN, READ, UPDATE, and WRITE

LIST=(*range-list*)

OPTION WITH SORT

$$\left\{ \begin{array}{l} \text{SORT=(}\textit{range-list}\text{)} \\ \text{SORT=(}\textit{item-name1}\left[\begin{array}{l}\text{(ASC)}\\\text{(DES)}\end{array}\right]\text{[,}\textit{item-name2}\left[\begin{array}{l}\text{(ASC)}\\\text{(DES)}\end{array}\right]\ \ldots\ \text{])} \end{array} \right\}$$

Example

```
FILE(CONTROL) MPEFILE,CODE=45;
```

FIND

FIND[(*modifier*)]*source*[,*option-list*];

MODIFIERS

```
none    - Retrieves from master set
CHAIN   - Retrieves from detail set or KSAM file
          based on key
CURRENT - Retrieves last entry accessed from KSAM file,
          data set. or MPE file
DIRECT  - Retrieves entry at specified record number
PRIMARY - Retrieves from master set at the primary location
RCHAIN  - Same as CHAIN only reverse order for data set;
          same as CHAIN for KSAM file
RSERIAL - Same as SERIAL only reverse order for data set;
          same as SERIAL for KSAM or MPE file
SERIAL  - Retrieves entries in serial mode
```

SOURCE

file-name
set-name[(*base-name*)]

OPTIONS

ERROR=*label*[([*item-name*])]
LIST=(*range-list*)
LOCK
NOMATCH
NOMSG
PERFORM=*label*
RECNO=*item-name*[(*subscript*)]
SINGLE
SOPT

SORT=(*item-name1* [(ASC) / (DES)] [,*item-name2* [(ASC) / (DES)] ...])

STATUS
WORKFILE

Example

```
FIND(SERIAL) CUST-M,LIST=(CUSTNUM),
             SORT=(CUSTNUM),PERFORM=PARA;
```

FORMAT

FORMAT *display-list*;

DISPLAY LIST
display-field[,*format-parm*]...
[:*display-field*[,*format-parm*]...]

DISPLAY FIELDS
item-name[(*subscript*)]
"*string*"
$CPU
$DATELINE
$PAGE
$TIME
$TODAY

FORMAT PARAMETERS
CCTL=*number*
CENTER
COL=*number*
EDIT="*edit-string*"
HEAD="*character-string*"
JOIN[=*number*]
LEFT
LINE[=*number*]
LNG=*number*
NEED=*number*
NOCRLF
NOHEAD
NOSIGN
PAGE[=*number*]
RIGHT
ROW=*number*
SPACE[=*number*]
TITLE
TRUNCATE
ZERO[E]S

Example

```
FORMAT CUSTNUM,LINE=2,NOHEAD,COL=10;
```

GET

GET[(*modifier*)]*source*[,*option-list*];

MODIFIERS

none - Retrieves from master set
CHAIN - Retrieves from detail set or KSAM file
based on key
CURRENT - Retrieves last entry accessed from KSAM or
MPE file or data set
DIRECT - Retrieves entry at specified record number
FORM - Displays VPLUS form and retrieves data
KEY - Calculated access on master set
PRIMARY - Retrieves from master set at the primary location
RCHAIN - Same as CHAIN in reverse order for data set;
same as CHAIN for KSAM file
RSERIAL - Same as SERIAL in reverse order for data set;
same as SERIAL for KSAM or MPE file
SERIAL - Retrieves entries in serial mode

SOURCE

file-name
set-name[(*base-name*)]

SOURCE WITH FORM

form-name
(*item-name*[(*subscript*)])
* - the CURRENT FORM name
& - the NEXT FORM name

OPTIONS WITHOUT THE FORM MODIFIER

ERROR=*label*([*item-name*])
LIST=(*range-list*)
LOCK
NOFIND
NOMATCH
NOMSG
RECNO=*item-name*[(*subscript*)]
STATUS

OPTIONS WITH THE FORM MODIFIER

APPEND
AUTOREAD
CLEAR
CURRENT
CURSOR=*field-name*|*item-name*[(*subscript*)]
FEDIT

FKEY=*item-name*[(*subscript*)]
Fn[(AUTOREAD)]=*label*
FREEZE
INIT
LIST=(*range-list*)
STATUS
WINDOW=([*field,*]*message*)

Examples

```
GET(DIRECT) ORDER-D,LIST=(CUSTNUM:QTY),RECNO=REC;

GET(FORM)FORM1,INIT,LIST=(),
WINDOW=(field1,"This field must be numeric");
```

GO TO

GO TO *label*;

Example

```
GO TO PARA1;
```

IF

IF *condition-clause* THEN *statement* [ELSE *statement*];

CONDITION CLAUSE
test-variable relational-operator value[,*value*...][{AND|OR} ...]

TEST VARIABLES
(*item-name*[(*subscript*)])
arithmetic-expression

```
EXCLAMATION  - Status of automatic null response
FIELD        - Status of the FIELD option
INPUT        - Last value input to INPUT
PRINT        - Status of PRINT or TPRINT option
REPEAT       - Status of REPEAT option
SORT         - Status of SORT option
STATUS       - Value of STATUS register
```

RELATIONAL OPERATORS

```
=
<>
<
<=
>
>=
```

When *test variable* is (*item-name*[(*subscript*)]) and *relational operator* is "=" or "<>", *value* can be:

```
NUMERIC
ALPHABETIC
ALPHABETIC-LOWER
ALPHABETIC-UPPER
```

Example

```
IF (QTY)>(ONHAND) THEN DISPLAY "Too many ordered!";
```

INPUT

INPUT "*prompt-string*"[,*option-list*];

OPTIONS
BLANKS
NOECHO
STATUS

Example

```
INPUT "Do you want the report on printer? (Y/N)";
```

LET

LET *destination-variable* = *arithmetic-expression*[,ERROR=*label*[([*item-name*])]];

DESTINATIONS
(*item-name*)
LINE - Value of the line counter
OFFSET(*item-name*) - The offset of a child item
PAGE - Value of the page counter
PLINE - Line counter for printer output
STATUS - Value of the status register
TLINE - Line counter for terminal output

ARITHMETIC EXPRESSION
[-]*source1* [*operator source2*]

OPERATORS
+
-
*
/
//
**

FUNCTIONS

ASCII(*item-name*|*value*)
LENGTH(*item-name*|*value*)
LN(*item-name*|*value*)
LOG(*item-name*|*value*)
POSITION(*item-name*|*value*, *item-name*|*value*);
SQRT(*item-name*|*value*)
VALUE(*item-name*|*value*)

Example

```
LET (ONHAND)=(ONHAND)-(QTY);

LET (S-LENGTH) = LENGTH(STRING);
```

LEVEL

LEVEL[(*label*([*item-name*]))];

Example

```
LEVEL(finished-level());
```

LIST

LIST[(*modifier*)] *item-name1*[, *option-list*][: *item-name2*[, *option-list*]] ... ;

MODIFIERS

```
none    - Adds item name to list register
AUTO    - Adds all file items to list register
KEY     - Adds item-name to key register
MATCH   - Adds item to list and match register
PATH    - Adds item to list and key register
UPDATE  - Adds item to list and update register
```

OPTIONS

```
ACCOUNT
ALIGN (All modifiers except KEY)
DATE
DATE/C
DATE/D
DATE/J
DATE/L
DATE/Y
GROUP
HOMEGROUP
INIT[IALIZE]
PASSWORD
PROCTIME
TERMID
TIME
TIMER
SESSION
USER
```

OPTIONS WITH MATCH

```
NE
LT
LE
GT
GE
LEADER
SCAN
TRAILER
```

OPTIONS WITH AUTO

@ - Lists all defined items

file-name - Name of file to list

Example

```
LIST CUSTNO,INIT: CUSTNAME,INIT: USERNAME,INIT;
```

LOGTRAN

LOGTRAN(*modifier*) *base*, *log-message*[,*option-list*];

MODIFIERS

```
BEGIN    - Writes a DBBEGIN log record
END      - Writes a DBEND log record
MEMO     - Writes a DBMEMO log record
XBEGIN   - Writes an XBEGIN Transaction Manager log record
           and starts a dynamic transaction
XEND     - Writes an XEND Transaction Manager log record
           and ends a dynamic transaction
XUNDO    - Writes an XUNDO Transaction Manager log record
           and undoes a dynamic transaction
```

OPTIONS

```
NOMSG
STATUS
```

OPTION WITH BEGIN, XBEGIN

LOCK(*setlist*)

Examples

```
LOGTRAN(BEGIN) CUSTDB,"Begin logging/lock database", LOCK(@);
LOGTRAN(XBEGIN) $HOME, "Begin dynamic txn logging", LOCK(@);
```

MOVE

MOVE (*item-name*)=*source-expression*;

SOURCE EXPRESSION

[-](*item-name*[(*subscript*)])
[-]"*character-string*"
[-]*string-function*
format-function
source1 {+|-} *source2* {+|-} ... *sourcen*
STATUS(DB)
STATUS(BASE)
STATUS(FILE)

FUNCTIONS

CHAR(*item-name*|*value*)
COL(*item-name*|"*character string*",*position*)
LOWER(*item-name*|"*character string*")
PROPER(*item-name*|"*character string*")
SPACE(*item-name*|"*character string*",*space-size*)
STRING(*item-name*|"*character string*",*position*,*length*)
UPPER(*item-name*|"*character string*")

Example

```
MOVE (FULLNAME)=(FNAME)+(LNAME);
```

OUTPUT

OUTPUT[(*modifier*)]*source*[,*option-list*];

MODIFIERS

```
none    - Retrieves from master set
CHAIN   - Retrieves from detail set or KSAM file
          based on key
CURRENT - Retrieves last entry accessed from MPE or KSAM
          file or data set
DIRECT  - Retrieves entry at specified record number
PRIMARY - Retrieves from master set at the primary location
RCHAIN  - Same as CHAIN in reverse order for data set;
          same as CHAIN for KSAM file
RSERIAL - Same as SERIAL in reverse order for data set;
          same as SERIAL for MPE or KSAM file
SERIAL  - Retrieves entries in serial mode
```

SOURCE
file-name
set-name[(*base-name*)]

OPTIONS
ERROR=*label*([(*item-name*)])
LIST=(*range-list*)
LOCK
NOCOUNT
NOHEAD
NOMATCH
NOMSG
PERFORM=*label*
RECNO=*item-name*[(*subscript*)]
SINGLE
SOPT

SORT=(*item-name* [(ASC) / (DES)] [,*item-name* ...])

STATUS

Example

```
OUTPUT(SERIAL) ORDER-D,LIST=(CUSTNO:QTY),NOMATCH;
```

PATH

PATH *source*[, *option-list*] ;

SOURCE
file-name
set-name[(*base-name*)]

OPTIONS
ERROR=*label*([*item-name*])
LIST=(*range-list*)
NOMSG
STATUS

Example

```
PATH ORDER-D,NOMSG;
```

PERFORM

PERFORM *label*;

Example

```
PERFORM PRINT-PARA;
```

PROC

PROC *procedure-name* [(*parameter-list*)] [, *option-list*] ;

PARAMETER LIST
(*item-name*[(*subscript*)])
ARG
ARGLNG
BASE[(*base-name*)]
BASELNG[(*base-name*)]
BYTE(*item-name*)
COUNT(*item-name*)
DECIMAL(*item-name*)
FILEID(*file-name*)
INPUT
INPUTLNG
ITEM(*item-name*)
ITEMLNG(*item-name*)
KEY
KEYLNG
POSITION(*item-name*)
SET(*set-name*)
SETLNG(*set-name*)
SIZE(*item-name*)
STATUS
STATUS(DB)
STATUS(IN)
TYPE(*item-name*)
VCOM(*form-file*)

PARAMETER PREFIXES
% - Passes by byte address
# - Passes by value
& - Copies function value of intrinsic to item

OPTIONS
COBOL|FORTRAN|BASIC|Pascal|SPL
NOTRAP
NOLOAD
UNLOAD

Examples

```
PROC DBCLOSE(BASE(CUST),SET(CUST-M),(MODE),STATUS(DB));
PROC CREATE(%(ROUTINE),,(CPIN),,#(CFLAG),,,,,,#(MAP));
```

PROMPT

PROMPT[(*modifier*)]*item-name*[("*prompt-string*")][,*option-list*]
[:*item-name*[("*prompt-string*")][,*option-list*]]...;

MODIFIERS

none	-	Places value in list and data registers
KEY	-	Places value in key and argument registers
MATCH	-	Places value in list, data, and match registers
PATH	-	Places value in list, data, key, and argument registers
SET	-	Places value in list and data registers if entered
UPDATE	-	Places value in list, data, and update registers

OPTIONS

ALIGN (All modifiers except KEY)
BLANKS
CHECK=*set-name*
CHECKNOT=*set-name*
NOECHO
RIGHT
STATUS

OPTIONS WITH MATCH

NE
LT
LE
GT
GE
LEADER
SCAN
TRAILER

Example

```
PROMPT CUSTNO ("Please enter customer number",)CHECK=CUST-M;
```

PUT

PUT[(*modifier*)]*destination*[,*option-list*];

MODIFIERS
none
FORM

DESTINATIONS
set-name[(*base-name*)]
file-name

DESTINATIONS WITH FORM MODIFIER
form-name
(*item-name*[(*subscript*)])
* - the CURRENT FORM name
& - the NEXT FORM name

OPTIONS
LIST=(*range-list*)
STATUS

OPTIONS WITHOUT THE FORM MODIFIER
ERROR=*label*([*item-name*])
LIST=(*range-list*)
LOCK
NOMSG
RECNO=*item-name*[(*subscript*)]
STATUS

OPTIONS WITH ONLY THE FORM MODIFIER
APPEND
CLEAR
CURRENT
CURSOR=*field-name*|*item-name*[(*subscript*)]
FEDIT
FKEY=*item-name*[(*subscript*)]
F*n*=*label*
FREEZE
INIT
LIST=(*range-list*)
STATUS
WAIT=[*Fn*]
WINDOW=([*field*,]*message*)

Example

```
PUT ORDER-D,LIST=(CUSTNO:QTY);
```

REPEAT

REPEAT *statement* UNTIL *condition-clause*;

CONDITION CLAUSE
test-variable relational-operator value[,*value*...][{AND|OR}...]

TEST VARIABLES
(*item-name*[(*subscript*)])
arithmetic-expression

```
EXCLAMATION - Status of automatic null response
FIELD       - Status of the FIELD option
INPUT       - Last value input to INPUT
PRINT       - Status of PRINT option
REPEAT      - Status of REPEAT option
SORT        - Status of SORT option
STATUS      - Value of status register
```

RELATIONAL OPERATORS

```
=
<>
<
<=
>
>=
```

When *test variable* is (*item-name*[(*subscript*)]) and *relational-operator* is "=" or "<>", *value* can be:

```
NUMERIC
ALPHABETIC
ALPHABETIC-LOWER
ALPHABETIC-UPPER
```

Example

```
REPEAT LET (A)=(A)/10 UNTIL (A)<=1;
```

REPLACE

REPLACE[(*modifier*)]*source*[,*option-list*] ;

MODIFIERS

```
none    - Updates master set entry
CHAIN   - Updates detail or KSAM chain based on key
CURRENT - Updates last entry accessed from file or
          data set
DIRECT  - Updates entry at specified record number
PRIMARY - Updates master set at primary location
RCHAIN  - Same as CHAIN in reverse order for data set;
          same as CHAIN for KSAM file
RSERIAL - Same as SERIAL in reverse order for data set;
          same as SERIAL for MPE or KSAM file
SERIAL  - Updates entries in serial mode
```

SOURCE

file-name
set-name[(*base-name*)]

OPTIONS

ERROR=*label*([*item-name*])
LIST=(*range-list*)
LOCK
NOCOUNT
NOMATCH
NOMSG
PERFORM=*label*
RECNO=*item-name*[(*subscript*)]
SINGLE
SOPT
STATUS
UPDATE

Example

```
REPLACE(SERIAL) ORDER-D,LIST=(CUSTNO:QTY),NOCOUNT;
```

RESET

RESET(*modifier*) [*target*];

MODIFIERS
COMMAND - Clears user response from buffer
DELIMITER - Resets delimiters to default
LANGUAGE - Resets language
OPTION - Resets various execution parameters
PROPER - Resets delimiters for upshifting the next letter
STACK - Resets list register pointer

TARGET WITH STACK MODIFIER
LIST

TARGETS WITH OPTION MODIFIER
AUTOLOAD
END
FIELD
FORMSTORE=(*form-store-list*)

MATCH [LIST({ [*item-name*] / * })]

NOHEAD
NOLOCK
NOLOOKAHEAD
PRINT
SORT
SUPPRESS
TPRINT

UPDATE [LIST({ [*item-name*] / * })]

VPLS

Example

```
RESET(OPTION) NOHEAD;
```

RETURN

RETURN[(*level*)];

Examples

```
RETURN;  RETURN(5);
```

SET

SET(*modifier*) *target*;

MODIFIERS

```
COMMAND   - Invokes command mode
DELIMITER - Changes delimiters
FORM      - Specifies transfer to VPLUS buffer
KEY       - Sets value of key and argument register
LANGUAGE  - Specifies language
MATCH     - Sets up match register
OPTION    - Sets execution parameters
PROPER    - Sets delimiters for upshifting the next letter
STACK     - Sets list register pointer
UPDATE    - Sets value of update register
```

TARGETS WITH COMMAND

EXIT
INITIALIZE
COMMAND[(*command-label*)]"*input-string*"

TARGET WITH DELIMITER,PROPER

"*delimiter-string*"

TARGET WITH STACK, UPDATE, MATCH, KEY, LANGUAGE

LIST(*item-name1*)

OPTIONS WITH MATCH MODIFIER

NE
LT
LE
GT
GE
LEADER
SCAN
TRAILER

TARGETS WITH FORM

form-name[,*option list*]
(*item-name*[(*subscript*)])[,*option-list*]
* [,*option-list*] - the CURRENT FORM name
& [,*option-list*] - the NEXT FORM name

OPTIONS WITH FORM MODIFIER

APPEND
CLEAR
CURSOR=*field-name*|*item-name*[(*subscript*)])
FEDIT
FREEZE
INIT
LIST=(*range-list*)
WINDOW=([*field*],*message*)

TARGETS WITH OPTION MODIFIER

AUTOLOAD
DEPTH=*number*
END=*label*
FIELD[="*ab*"]
FORMSTORE=(*form-store-list*)
HEAD
LEFT
NOBANNER
NOHEAD
NOLOCK
NOLOOKAHEAD
PALIGN=*number*
PDEPTH=*number*
PRINT
PROMPT=*number*
PWIDTH=*number*
REPEAT
RIGHT
SORT
SUPPRESS
TABLE
TPRINT
VPLS=*item-name*[(*subscript*)]
WIDTH=*number*
ZERO[E]S

Examples

```
SET(OPTION) PRINT;

SET(FORM) LIST-FORM,LIST=(LIST-DATE),
    WINDOW=(LIST-DATE,"Only enter orders for this date");
```

SYSTEM

```
SYSTEM program-name[,definition-list];

DEFINITIONS
BANNER="text"
BASE=base-name1[(["password"][,mode]
                [,optlock][,[basetype]]]])]

   [,base-name2[(["password"][,mode]
                [,optlock][,[basetype]]]])]]...
DATA=data-length,data-count
FILE=file-name1[(access[(file-option-list)]
               [,rec-length[,blocking-factor
               [,file-size[,extents[,initial-allocation
               [,file-code]]]]]])]...
FSTORESIZE=formstoresize
KSAM=file-name1[(access[(file-option-list)])]...
OPTION={TEST|NOTEST}
SIGNON="text"
SORT=number
VPLS=file-name1[(form-name1[(item-list1)] ...)] ...] ...
WORK=work-length,work-count
WORKFILE=number
```

Example

```
SYSTEM PROG1, BASE=DB1("PASS",1);
```

UPDATE

UPDATE[(FORM)] *destination*[,*option-list*];

DESTINATIONS
file-name
set-name[(*base-name*)]

DESTINATIONS WITH FORM
form-name
(*item-name*[(*subscript*)])
* - the CURRENT FORM name
& - the NEXT FORM name

OPTIONS
ERROR=*label*([*item-name*])
LIST=(*range-list*)
LOCK
NOMSG
STATUS

OPTIONS WITH FORM
APPEND
CLEAR
CURSOR=*field-name*|*item-name*[(*subscript*)]
FEDIT
FKEY=*item-name*[(*subscript*)]
Fn=*label*
FREEZE
INIT
LIST=(*range-list*)
STATUS
WAIT=[*Fn*]
WINDOW=([*field,*]*message*)

Examples

```
UPDATE ORDER-D,LIST=(DATE);

UPDATE(FORM) *,WINDOW=((FIELD-ENH),(WINDOW-MSG));
```

WHILE

WHILE *condition-clause statement*;

CONDITION CLAUSE
test-variable relational-operator value[,*value*...][{AND|OR}...]

TEST VARIABLES
(*item-name*[(*subscript*)])
arithmetic-expression

```
EXCLAMATION  - Status of automatic null response
FIELD        - Status of the FIELD option
INPUT        - Last value input to INPUT
PRINT        - Status of PRINT option
REPEAT       - Status of REPEAT option
SORT         - Status of SORT option
STATUS       - Value of STATUS register
```

RELATIONAL OPERATORS

```
=
<>
<
<=
>
>=
```

When test variable is (*item-name*[(*subscript*)]) and *relational-operator* is "=" or "<>", *value* can be:

```
NUMERIC
ALPHABETIC
ALPHABETIC-LOWER
ALPHABETIC-UPPER
```

Example

```
WHILE (A)>10 LET (A)=(A)/10;
```

2

Running Transact

Compiler Output Commands

!COPYRIGHT("*text-string*")
!INCLUDE(*file-name*)
!LIST
!NOLIST
!PAGE
!SEGMENT[("*text-string*")]

System Dictionary Commands

!SYSDIC[(*dictionary.group.account*)]
!NOSYSDIC
!DOMAIN [(*domain*)]
!VERSIONSTATUS[(P/T/A)]
!VERSION[(version)]
!SCOPE[(scope[,"password"])]

Conditional Compile Commands

!SET X*n* {ON/OFF}
!IF X*n*={ON/OFF}
!ELSE
!ENDIF

!IF XL={ON/OFF}
!ELSE
!ENDIF

Transact/V Program Compilation

```
:RUN TRANCOMP.PUB.SYS

SOURCE
FILE>        Enter the source file name

LIST FILE>   <cr>       to send listing to $STDLIST
             LP         to send listing to TRANLIST
             NULL       to suppress compiler listing
             *file-name to back reference a file equation
             file-name  to send listing to a disk file

CONTROL>     LIST       Generates a listing of the compiled source code
             DICT       References the DICTIONARY to resolve item definitions
             CODE       Creates p-code if no errors occur
             ERRS       Lists errors on $STDLIST even if you redirect listing
             CHCK       Produces a list of items that are used but never put on
                        the list register
             DEFN       Produces a list of item definitions
             OBJT       Produces a listing of the p-code
             OPT@       Optimizes compilation tables and IP file
             OPTE       Eliminates edit mask specs from IP file
             OPTH       Eliminates heading text from IP file
             OPTI       Eliminates item names from IP file
             OPTP       Eliminates prompt strings from IP file
             OPTS       Optimizes segmented Transact programs
             STAT       Generates statistics on data stack usage
             XERR       Creates p-code file even if errors are encountered
             XREF       Provides a cross reference listing of labels and
                        references

DEFAULTS:    LIST,DICT,CODE,ERRS
```

Note

You can precede any of the above CONTROL options with "NO" to reverse its effect. For example, `NOXREF`.

Formal File Designators for Transact/V Compilation

TRANCODE	Name of the file opened and used by the compiler. Default size is 1023 records.
TRANIN	Formal file designator for responses to prompts issued by compiler. Default is `$STDINX`.
TRANLIST	Formal file designator for destination of compiler listings when LP is the response to the `LIST FILE` prompt.
TRANOUT	Formal file designator for output from the compiler that by default, is sent to the standard list device.
TRANTEXT	Formal file designator for the source code file.

Transact/V Program Execution

```
:RUN TRANSACT.PUB.SYS

SYSTEM NAME> program-name[,mode[,test-parm[,start-location[,end-location]]]]
```

Formal File Designators for Transact/V Execution

TRANDUMP	Formal file designator for the destination of test mode output.
TRANIN	Formal file designator for responses issued by the processor. The default is `$STDINX`.
TRANLIST	Formal file designator for output from the processor that is normally sent to the line printer.
TRANOUT	Formal file designator for output from the processor that is normally sent to the standard list device.
TRANSORT	Name of the sort file opened and used by the processor in sort operations. Default size is 10,000 records.
TRANVPLS	Name of the file used by the processor to open the VPLUS terminal.

Transact/iX Compilation and Execution

```
TRANXL [textfile] [,[rlfile] [,[listfile]]] [;INFO="compiler-options"]

RUN TRAN.PUB.SYS [;PARM=parmnum] [;INFO="compiler-options"]

LINKEDIT

LINK [FROM=file[,file]...] [;TO=destfile]
     [;RL=rlfile]
     [;XL=xlfile]
     [;CAP=caplist]
     [;STACK=maxstacksize]
     [;HEAP=maxheapsize]
     [;UNSAT=unsatname]
     [;PARMCHECK=integer]
     [;PRIVLEV=integer]
     [;XLEAST=integer]
     [;ENTRY=entryname]
     [;NODEBUG]
     [;NOSYM]
     [;MAP]
     [;SHOW]

TRANXLLK [textfile,[progfile] [,[listfile]]] [;INFO="compiler-options"]

TRANXLGO [textfile][,[listfile]][;INFO="compiler-options"]

RUN progname [;XL="xlname[,xlname,...]"]
```

Transact/iX Compiler Options

```
CHCK
CODE_OFFSETS
DEFN
DYNAMIC_CALLS
ERRS
HP3000_16
LIST
OPTE
OPTH
OPTI
OPTIMIZE
OPTP
OPTS
OPT@
```

```
PROCALIGNED_16
PROCALIGNED_32
PROCALIGNED_64
PROCINTRINSIC
SUBPROGRAM
TRANDEBUG
XREF

DEFAULTS:
NODYNAMIC_CALLS
NOHP3000_16
NOPROCALIGNED
NOPROCINTRINSIC
NOOPTIMIZE
```

Note

You can precede any of the above options with "NO" to reverse its effect. For example, `NOXREF`.

Formal File Designators for Transact/iX Compilation

TRANLIST	Formal file designator for output from the compiler that, by default, is sent to the standard list device. The default is `$STDLIST`.
TRANOBJ	Formal file designator of the RSOM file. The default is `$NEWPASS`.
TRANTEXT	Formal file designator for the source code file. The default is `$STDIN`.

Formal File Designators for Transact/iX Execution

TDBGINIT	Formal file designator that is checked for during start up of TRANDEBUG. If it is present, it is treated as a user file.
TDBUGIN	Formal file designator where TRANDEBUG reads user input information. It is typically assigned to the `$STDINX`.
TDBUGOUT	Formal file designator where TRANDEBUG responds to user's requests. It is typically assigned to the `$STDLST`.
TRANIN	Formal file designator for responses issued by the processor. The default is `$STDINX`.
TRANLIST	Formal file designator for output from the processor that is normally sent to the line printer.
TRANOUT	Formal file designator for output from the processor that is normally sent to the standard list device.
TRANSORT	Name of the sort file opened and used by the processor in sort operations. Default size is 10,000 records.

TRANVPLS Name of the file used by the processor to open the VPLUS terminal.

Architected Call Interface (ACI)

`TL_CALL_TRANSACT` (*program_name, data_buffer, data_length, return_status*)

program_name Name of Transact/iX subprogram to call. Must be terminated with a blank.

data_buffer User defined structure for passing data to and from subprogram.

data_length Maximum data register size (in bytes) in subprogram.

return_status Status value indicating success or failure of the intrinsic call.

3

Test Facility (Transact/V)

Enabling Test Modes

```
RUN TRANSACT.PUB.SYS

SYSTEM NAME> PROG1,,test-parameter[,[segment1.]starting-instru ction-address]
             [,[segment2.]ending-instruction-address]

       or  Ctrl Y

           >TEST [test-parameter [,[segment1.]starting-instruction-address
               [,[segment2.]ending-instruction-address]]
```

Example

```
TEST 44,0,333
```

Test Parameters

PARM	FUNCTION
none	Switches off existing test mode.
1	Displays information about the file or database operation if error occurs.
2	Displays each instruction address and compiler code at that address.
3	Displays each instruction address, compiler code, space used by list and data registers, and amount of remaining processor work space.
4	Displays each instruction address, compiler code, instruction timings, and HP3000 data stack pointers Z, S, Q, and DL.
22	Displays each instruction address, compiler code, and key and argument register values used by the database or file operation.
23	Displays each instruction address, compiler code, and contents of list, data, key, argument, match and update registers if used by the database or file operation.
24	Same as 23 except it is only issued when an accessed record meets the selection criteria in the match register.
25	Same as 24 except display is issued for every record accessed by the database or file operation.

34	Displays the instruction address, compiler code, and contents of the VPLUS buffer following an instruction generated by a statement that references a VPLUS form.
42	Displays the instruction address and compiler code only if the instruction is not listed in the compiler listing. Displays contents of the list and data registers whenever the contents of list register changes.
43	Displays the instruction address, compiler code, and the contents of the list and data registers for every instruction.
44	Displays the instruction address, compiler code, and the contents of all registers for every instruction.
101	Lists the data and workspace recovery statistics for every command sequence.
102	Lists the data and workspace recovery statistics for the entire program.
121	Issues an overlay trace whenever a program switches segments.
122	Issues a trace whenever a file is locked or unlocked.
123	Issues a workspace recovery message whenever recovery is needed.

4

TRANDEBUG Commands (Transact/iX)

:

Allows access to the MPE/iX Command Interpreter.

```
: [COMMAND]
```

ABORT

Terminates execution of the Transact/iX program and exits TRANDEBUG.

```
ABORT
```

AUTORPT

Allows you to repeat the last command typed by pressing the Return key.

```
{AUTORPT} [ON ]
{AR     } [OFF]
```

BREAK DELETE

Deletes a specified breakpoint.

```
{BREAK DELETE} [#breakpoint_number                                  ]
{BD          } [p-code_offset [ ,segment [ ,system] ]               ]
               [ALL                                                 ]
```

BREAK LIST

Lists the breakpoints that have been set in the Transact/iX program.

```
{BREAK LIST}
{BL        }
```

BREAK SET

Sets a breakpoint at the specified location.

```
{ BREAK SET } { offset [ ,segment] [ ,system ] }
{ BS        } { system                         } [ ,count] [ ,{cmdlist} ]
```

CONTINUE

Continues execution of the Transact/iX program until a breakpoint is encountered or the program completes execution.

```
{ CONTINUE }
{ C        }
```

DATA BREAK DELETE

Deletes a specified data breakpoint.

```
{ DATA BREAK DELETE } [ #breakpoint_number                    ]
{ DBD              } [ [ item_name     ] [ ,system ]         ]
                     [ [ register_name ]                     ]
                     [ ALL                                   ]
```

DATA BREAK LIST

Lists the data breakpoints currently set in the Transact/iX program.

```
{ DATA BREAK LIST }
{ DBL             }
```

DATA BREAK REGISTER

Sets a breakpoint at the specified register.

```
{ DATA BREAK REGISTER } register [ ,[ count] [ ,{ cmdlist } ] ]
{ DBR                }
```

DATA BREAK SET

Sets a breakpoint at the specified location.

```
{ DATA BREAK SET } item_name [ [ ,length[ ,count [ ,{ cmdlist } ] ] ] ]
{ DBS           }            [ ,rel-op ,value [ ,{ cmdlist } ]     ]
```

DATA LOG

Allows logging of TRANDEBUG commands and output to a file.

```
{ DATA LOG } [ filename ]
{ DL       } [ ON       ]
             [ OFF      ]
             [ CLOSE    ]
```

DEFN

Displays information about the definition of a particular item.

```
DEFN item_name
```

DISPLAY BASE

Displays information for specific databases.

```
{ DISPLAY BASE } [ base_name ]
{ DB           }
```

DISPLAY CALLS

Displays the CALL stack from the currently executing system back to the main program.

```
{ DISPLAY CALLS }
{ DCA           }
```

DISPLAY COMAREA

Displays the contents of the currently active VPLUS communication area (comarea).

```
{ DISPLAY COMAREA }
{ DCO             }
```

DISPLAY FILE

Displays information for specific MPE/KSAM files.

```
{ DISPLAY FILE } [ file_name ]
{ DF           }
```

DISPLAY INPUT

Displays the contents of the input register.

```
{ DISPLAY INPUT }
{ DIN           }
```

DISPLAY ITEM

Displays the value of an item or all items in the data register.

```
{ DISPLAY ITEM } [ item_name_list ]
{ DIT          }
```

item_name_list =

```
{ item1 [ (subscript) ] : item2 [ (subscript) ]                    }
{ item1 [ (subscript) ] [ , ... itemN [ (subscript) ] ]            }
{ item1 [ (subscript) ] :                                          }
{ : item1 [ (subscript) ]                                          }
```

Note that child items cannot be specified for range boundaries.

DISPLAY KEY

Displays the item in the key register and the corresponding value in the argument register.

```
{ DISPLAY KEY }
{ DK          }
```

DISPLAY MATCH

Displays the contents of the match register.

```
{DISPLAY MATCH}
{DM           }
```

DISPLAY PERFORMS

Displays the current PERFORM stack.

```
{DISPLAY PERFORMS}[ALL]
{DP              }
```

DISPLAY STATUS

Displays the value in the status register.

```
{DISPLAY STATUS}
{DS            }
```

DISPLAY STATUSDB

Displays the contents of the database status array returned by the last TurboIMAGE/XL call.

```
{DISPLAY STATUSDB}
{DSDB            }
```

DISPLAY STATUSIN

Displays the value in the statusin register.

```
{DISPLAY STATUSIN}
{DSIN            }
```

DISPLAY UPDATE

Displays the specified item in the update register.

```
{DISPLAY UPDATE}
{DU            }
```

EDIT

Invokes HP EDIT from within TRANDEBUG, allowing you to edit or browse source files.

```
EDIT [filename]
```

HELP

Provides online assistance for TRANDEBUG.

```
{HELP} [ [ command [ [ALL     ] ] ]
{?   }                [PARMS   ]
                      [EXAMPLES]
       [ HELPINSTRUCTIONS       ]
```

LABEL BREAK SET

Sets a breakpoint at the specified label.

```
{LABEL BREAK SET} label[ ,segment[ ,count[ ,{ cmdlist } ] ] ]
{LBS            }
```

LABEL JUMP

Moves the source code window to a specific label in the code.

```
{LABEL JUMP} label[ ,segment]
{LJ        }
```

LOC

Indicates the p-code offset, segment, and system of the Transact/iX statement that executes next.

```
LOC
```

LOG

Allows logging of TRANDEBUG commands to an MPE/iX file.

```
LOG [ filename ]
    [ ON       ]
    [ OFF      ]
    [ CLOSE    ]
```

MODIFY INPUT

Modifies the contents of the input register.

```
{ MODIFY INPUT } [ new_value ]
{ MIN          }
```

MODIFY ITEM

Modifies the value of a data item in the data register.

```
{ MODIFY ITEM } item_name [ (subscript) ] [ new_value ]
{ MIT         }
```

MODIFY KEY

Modifies the item in the key register.

```
{ MODIFY KEY } [ new_item ]
{ MK         }
```

MODIFY MATCH

Modifies the specified item in the match register.

```
{ MODIFY MATCH } item_name [ new_value ]
{ MM          }
```

MODIFY STATUS

Modifies the value in the status register.

```
{ MODIFY STATUS } [ new_value ]
{ MS           }
```

MODIFY UPDATE

Modifies the specified item in the update register.

```
{ MODIFY UPDATE } item_name [ (subscript) ] [ new_value ]
{ MU            }
```

NMDEBUG

Transfers control from TRANDEBUG to NMDEBUG.

```
{ NMDEBUG }
{ NM      }
```

PAGE BACK

Pages back the source code window in the source file.

```
{ PAGE BACK }
{ PB        }
```

PAGE FORWARD

Pages the source code window forward in the source file.

```
{ PAGE FORWARD }
{ PF           }
```

PAGE JUMP

Moves the source code window to a specified segment and offset in the program.

```
{ PAGE JUMP } offset [ ,segment ]
{ PJ        }
```

PRINT

Alters or displays the Transact/iX PRINT option.

```
PRINT [ ON  ]
      [ OFF ]
```

REPEAT

Alters or displays the Transact/iX REPEAT option.

```
REPEAT [ ON  ]
       [ OFF ]
```

SORT

Alters or displays the Transact/iX SORT option.

```
SORT [ ON  ]
     [ OFF ]
```

STEP

Continues execution of the Transact/iX program for the specified number of statements.

```
{ STEP } [ number_of_steps ]
{ S    }
```

TPRINT

Alters or displays the Transact/iX TPRINT flag to format the output generated from the DISPLAY or OUTPUT verbs for printing.

```
TPRINT [ ON  ]
       [ OFF ]
```

TRACE

Turns the trace flag on or off for the specified type of trace.

```
{ TRACE } { CODE  } [ ON  ]
{ TR    } { IMAGE }  [ OFF ]
          { MPE   }
```

USE

Reads TRANDEBUG commands from the specified MPE/iX file.

```
USE filename
```

VERSION

Outputs the current version of the Transact/iX run time library.

```
VERSION
```

WINDOW LENGTH

Adjusts the size of the source code window.

```
{ WINDOW LENGTH } new_size
{ WL            }
```

WINDOW OFF

Turns the source code window off.

```
{ WINDOW OFF }
{ WOFF       }
```

WINDOW ON

Turns on the source code window.

```
{ WINDOW ON }
{ WON       }
```

TRANDEBUG Files

DATA LOG — The DATA LOG file is established and controlled via the DATA LOG command. The file records the user input from TDBUGIN as well as the output to TDBUGOUT. The actual file name for this file can be found in *data_log_file_name*.

LOG — The LOG file is established and controlled via the LOG command. The file records the user input from TDBUGIN so that it can be used later as a use file. The actual file name for this file can be found in *logfile_name*.

USE — The USE file can be assigned via a USE command. TRANDEBUG then uses this file as the input instead of TDBUBIN until an end of file is found. The file id for this file can be found in *use_file_number*.