WANG

VS

Procedure Language Quick Reference

VS Procedure Language Quick Reference

1st Edition — August, 1982
Copyright © Wang Laboratories, Inc., 1982
800-6201PP-03

WANG

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 ● TEL: 617/459-5000, TWX 710-343-6769, TELEX 94-7421

This quick reference replaces the *VS Procedure Language Pocket Guide* (800-6201PP-02).

WANG

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL: 617/459-5000, TWX 710-343-6769, TELEX 94-7421

INTRODUCTION

This quick reference is a guide for the users of Wang VS Procedure language and is intended for those already familiar with VS Procedure language. For a detailed discussion of the language, refer to the *VS Procedure Language Reference* (800-1205PR).

This reference defines the various VS Procedure language statements, diagrams the correct syntax and includes an example for each. It also presents a glossary of terms used in the VS Procedure language and lists the Procedure language return code values and their meanings.

The user of this document should be familiar with the concepts discussed in the *VS Programmer's Introduction* (800-1101PI) and the *VS Program Development Tools* (800-1307PT).

TABLE OF CONTENTS

PROCEDURE LANGUAGE SYNTAX

This section contains the general format and syntax of each Procedure language statement, including a description of the purpose and function of each. Procedure language syntax is generally free-form, but subject to the following rules:

1. Each procedure *must* contain a line starting with PROCEDURE or PROC.

2. Comment lines within the procedure must begin with an asterisk (*) in column 1, or be enclosed in paired square [ ] brackets.

3. A colon (:) must separate a statement label from the verb which follows. There can be no space between the label and the colon.

4. Multiple spaces between procedure elements are ignored by the Procedure Interpreter. Blank lines are allowed between statements or comments.

5. A procedure statement can be extended onto more than one line, if necessary, *without* special continuation characters.

6. Both uppercase and lowercase text can be used within a procedure. However, lowercase text is automatically converted to uppercase, except when such text is part of a constant enclosed in quotes.

7. The character in column 71 of line n is adjacent to the character in column 1 of line n+1.

8. All entries in column 72 are ignored.

The procedure verbs, with their syntax descriptions, are arranged in alphabetical order in this section for ease of reference. The syntax of each procedure conforms to the following format:

Capitalized words	=	Keywords
Lowercase words	=	Terms
, : = () + - ; !! & ' " .	=	Required syntax
! !	=	One item must be encoded
[ ]	=	Optional item
. . .	=	Preceding item may be repeated

In this guide, all statement examples that can be preceded by a label are given the label name of Test for consistency. In actual practice, any name that meets Procedure language naming conventions is permissible.

ASSIGN

The ASSIGN statement assigns a value to a variable or a variable substring.

General Format:

```
                      ( integer-variable )     ( integer-operand )
[label:] ... ASSIGN   { string-variable  }  =  { string-operand  }
                      ( substring        )

where:

                        (          ( integer-variable  ) )  [ ( + )  ( integer-variable ) ]
  integer-operand  =    { [ + ]    { integer-constant  } }  [ {   }  { integer-constant } ] ...
                        ( [ - ]    ( step-label        ) )  [ ( - )  ( step-label       ) ]

                           ( string-variable )   [      ( string-variable )  ]
  string-operand   =       { string-constant }   [ !!   { string-constant }  ] ...
                           { (refkey)        }   [      { (refkey)        }  ]
                           ( substring       )   [      ( substring       )  ]

  substring        =       string-variable  ( start [ , { length } ] )
                                                        {   *    }
```

Example:
Test: ASSIGN &VOLUME=VOL444

DECLARE

The DECLARE statement defines variables that are to be used within the procedure. It also specifies the types of variables and, optionally, their initial values.

General Format:

```
                                                  ( STRING (n)* )  [          ( string-constant  ) ]
[label:] ... DECLARE variable [, variable] ... [AS] {             }  [ INITIAL  {                  } ]
                                                  ( INTEGER     )  [          ( integer-constant ) ]
```

Example:
Test: DECLARE &FILE, &LIBRARY AS STRING (8)

* The value (n) is an integer between 1 and 256, inclusive.

DISMOUNT

The DISMOUNT statement logically dismounts a disk or tape volume. It is analogous to the DISMOUNT command issued through the Command Processor.

Format 1:

```
[label:] ... DISMOUNT [DISK] volname
```

Format 2:

```
[label:] ... DISMOUNT TAPE volname
```

Example:
Test: DISMOUNT TAPE VOL1

DISPLAY

The DISPLAY statement overrides current default values for a GETPARM request and displays a prompt enabling the user to supply values at runtime. DISPLAY is always part of the procedure step defined by the RUN statement it follows.

General Format:

$$\text{[label:] ... DISPLAY} \begin{Bmatrix}\text{prname}\\ \text{inner-label}\end{Bmatrix} \left[\begin{array}{l}\text{key1} = \begin{Bmatrix}\text{value1}\\ \text{(refkey1)}\end{Bmatrix} \left[,\ \text{key2} = \begin{Bmatrix}\text{value2}\\ \text{(refkey2)}\end{Bmatrix}\right] \ldots \\ \text{(spec-label)}\end{array}\right]$$

Example:
Test: DISPLAY INPUT FILE=XYZ, LIBRARY=#ABCLIB, VOLUME=VOL444

ENTER

The ENTER statement overrides current default values for a GETPARM request without generating a workstation transaction. ENTER is always part of the procedure step defined by the RUN statement it follows.

General Format:

```
                  ┌ prname      ┐                  ┌                ┌ value1    ┐   ┌         ┌ value2    ┐ ┐      ┐
[label:] ... ENTER┤             ├ [pfkey] [,]      │ key1  =        ┤           ├   │, key2 = ┤           ├ │ ...  │
                  └ inner-label ┘                  │                └ (refkey1) ┘   └         └ (refkey2) ┘ ┘      │
                                                   │ (spec-label)                                                  │
                                                   └                                                               ┘
```

Example:
Test: ENTER OUTPUT FILE=OUTFILE

EXTRACT

The EXTRACT statement retrieves information from the system and stores this information in variables. EXTRACT Format 1 is analogous to the EXTRACT SVC executed with the specified keywords. In Format 2, if IN is not specified, OUTLIB is assumed; if ON is not specified, OUTVOL is assumed. In Format 1, the keywords identifying fields from which data can be extracted are as follows:

CURLIB	INLIB	OUTVOL	PROGVOL	SPOOLVOL	TASK#	WORKLIB
CURVOL	INVOL	PRINTER	PRTCLASS	SYSLIB	TASKTYPE	WORKVOL
DISKIO	LINES	PRINTIO	RUNLIB	SYSVOL	USERID	WS
FILECLAS	OTIO	PRNTMODE	RUNVOL	SYSWORK	USERNAME	WSIO
FORM#	OUTLIB	PROGLIB	SPOOLIB	TAPEIO	VERSION	

Format 1:

```
[label:] ... EXTRACT  variable = key1  [, variable = key2] ...
```

Example:
Test: EXTRACT &MYNAME=&USERNAME, &WS=WS

EXTRACT (continued)

Format 2:

```
[label:] ... EXTRACT  integer-variable = {BLOCKS ALLOCATED FOR}
                                         {RECORDS USED BY     }

                                         {filename  [IN libname]  [ON volname]}
                                         {(spec-label)                        }
```

Example:
Test: EXTRACT &BLKS=BLOCKS ALLOCATED FOR FILE123

GOTO

GOTO can be a stand-alone statement or an IF statement clause. As a stand-alone statement, GOTO performs an unconditional branch to a specified statement. When part of an IF statement, GOTO performs a branch conditioned by the result of the IF test. Because multiple statements can have the same label, the following rules determine the target statement of the branch:

1. The first occurrence of a label following the GOTO statement in the procedure text is the target.

2. If no label exists, the closest occurrence of the label preceding the GOTO statement in the procedure is the target.

General Format:

```
[label:] ... GOTO {step-label}
                  {stmt-label}
```

Example:
Test: GOTO LABEL2

IF

The IF statement compares two values. If the result is true, then the following RETURN or GOTO statement is executed; otherwise it is ignored. The IF EXISTS statement checks for the existence of a file, library, or volume.

Format 1:

```
                                             ( EQ  )
                                             ( NEQ )
                                             ( LT  )
                                             ( NE  )
                                             ( NLT )
                                             ( GT  )
                  { integer-variable }       ( NGT )   { integer-variable }
[label:] ... IF   { integer-constant }       ( LE  )   { integer-constant }
                  { step-label       }       ( GE  )   { step-label       }
                                             ( <   )
                                             ( <=  )
                                             ( >   )
                                             ( >=  )
                                             ( <>  )
                                             ( =   )

     { GOTO     { step-label }                                                                        }
     {          { stmt-label }                                                                        }
     {                                                                                                }
     { RETURN [ CODE  -  { integer-variable }   [ ,  { integer-variable } ] ... } ]                   }
     {        [          { integer-constant }   [    { integer-constant } ]     } ]                   }
     {        [          { step-label       }   [    { step-label       } ]     } ]                   }
```

Example:
Test: IF &COUNTER > 100 GOTO LOOP

IF (continued)

Format 2:

```
                                          ⎧ EQ  ⎫
                                          ⎪ NEQ ⎪
                                          ⎪ LT  ⎪
                                          ⎪ NE  ⎪
                  ⎧ string-variable ⎫     ⎪ NLT ⎪   ⎧ string-variable ⎫
[label:] ... IF   ⎨ string-constant ⎬     ⎪ GT  ⎪   ⎨ string-constant ⎬
                  ⎪ substring       ⎪     ⎨ NGT ⎬   ⎪ substring       ⎪
                  ⎩ (refkey)        ⎭     ⎪ LE  ⎪   ⎩ (refkey)        ⎭
                                          ⎪ GE  ⎪
                                          ⎪ <   ⎪
                                          ⎪ <=  ⎪
                                          ⎪ >   ⎪
                                          ⎪ >=  ⎪
                                          ⎪ <>  ⎪
                                          ⎩ =   ⎭

   ⎧ GOTO     ⎧ step-label ⎫                                                                    ⎫
   ⎪          ⎩ stmt-label ⎭                                                                    ⎪
   ⎨                                                                                            ⎬
   ⎪ RETURN  ⎡ CODE  =  ⎧ integer-variable   ⎡ +  ⎧ integer-variable ⎫⎤ ...  ⎫⎤               ⎪
   ⎩         ⎣          ⎨ integer-constant   ⎢    ⎨ integer-constant ⎬⎥      ⎬⎦               ⎭
                        ⎩ step-label         ⎣    ⎩ step-label       ⎭⎦      ⎭
```

Example:
Test: IF &ID=&USERLIST (&INDEX,3) GOTO OK

IF (continued)

Format 3:

```
                                     ⎧⎧filename⎫                 ⎧libname⎫      ⎧volname⎫⎫
[label:] ... IF [NOT] EXISTS FILE    ⎨⎩(refkey)⎭        IN       ⎩(refkey)⎭ ON  ⎩(refkey)⎭⎬
                                     ⎩ (spec-label)                                      ⎭

                                     ⎧⎧libname ⎫                 ⎧volname⎫⎫
                           LIBRARY   ⎨⎩(refkey)⎭        ON       ⎩(refkey)⎭⎬
                                     ⎩ (spec-label)                        ⎭

                                     ⎧ volname     ⎫
                           VOLUME    ⎨ (refkey)    ⎬
                                     ⎩ (spec-label)⎭

        ⎧ GOTO     ⎧step-label⎫                                                               ⎫
        ⎪          ⎩stmt-label⎭                                                               ⎪
        ⎨                                                                                     ⎬
        ⎪ RETURN  ⎡CODE  =  ⎧integer-variable   ⎡   ⎧integer-variable ⎫⎤    ⎫⎤                ⎪
        ⎩         ⎣         ⎨integer-constant   ⎢ + ⎨integer-constant ⎬⎥ ...⎬⎦                ⎭
                            ⎩step-label         ⎣   ⎩step-label       ⎭⎦    ⎭
```

Example:
Test: IF EXISTS LIBRARY @SYSTEM@ ON VOL444 GOTO LABEL10

LOGOFF

LOGOFF terminates the user's current session. All programs and procedures initiated by the current RUN command are terminated, their files are closed, and a Command Processor LOGOFF command is issued. When programs or procedures are invoked from the EDITOR, the LOGOFF statement returns control to the invoking utility rather than cancelling the user's session.

General Format:

```
[label:]  ...  LOGOFF
```

Example:
Test: LOGOFF

MESSAGE

MESSAGE displays text on a workstation and resumes execution of the procedure. A semicolon (;) marks the end of a text line.

A constant or variable can optionally be preceded by one or more of the following attributes: UPPER, UPLOW, NUMERIC, BRIGHT, DIM, BLINK, BLANK, or LINE. The user should note that UPPER, UPLOW, and NUMERIC are used only with variables that are modifiable. If CENTER is specified for a line, the text on that line is horizontally centered. Otherwise, text lines are left justified beginning in column 2.

General Format:

[label:] ... MESSAGE

$$\left\{ [\text{CENTER}] \left[[\text{attribute}] \left\{ \begin{array}{l} \text{string-constant} \\ \text{integer-constant} \\ \text{string-variable} \\ \text{integer-variable} \end{array} \right\} \left[, [\text{attribute}] \left\{ \begin{array}{l} \text{string-constant} \\ \text{integer-constant} \\ \text{string-variable} \\ \text{integer-variable} \end{array} \right\} \right] \dots \right] ; \right\} \dots$$

$$[\text{CENTER}] \left[[\text{attribute}] \left\{ \begin{array}{l} \text{string-constant} \\ \text{integer-constant} \\ \text{string-variable} \\ \text{integer-variable} \end{array} \right\} \left[, [\text{attribute}] \left\{ \begin{array}{l} \text{string-constant} \\ \text{integer-constant} \\ \text{string-variable} \\ \text{integer-variable} \end{array} \right\} \right] \dots \right]$$

Example:

```
Test:   MESSAGE CENTER "STEP 5 HAS COMPLETED"
```

MOUNT

MOUNT is analogous to the MOUNT option of the Manage DEVICEs command on the Command Processor. It logically mounts a disk or tape. Volume type is not optional for disk volumes named DISK or tape volumes named TAPE.

General Format:

```
                                                   ⎡         ⎧STANDARD⎫ ⎧LABEL ⎫⎤
[label:] ... MOUNT ⎡DISK⎤ volname ON unit#          ⎢ [WITH] ⎨IBM     ⎬ ⎨      ⎬⎥
                   ⎣TAPE⎦                           ⎢        ⎨ANSI    ⎬ ⎩LABELS⎭⎥
                                                   ⎣         ⎩NO      ⎭         ⎦

                        ⎡       ⎧SHARED                  ⎫        ⎤
                        ⎢ [FOR] ⎨EXCLUSIVE               ⎬  USAGE ⎥
                        ⎢       ⎨PROTECTED               ⎬        ⎥
                        ⎣       ⎩RESTRICTED  [REMOVAL]   ⎭        ⎦
```

Example:
Test: MOUNT TAPE VOL1 ON 28 WITH NO LABEL FOR EXCLUSIVE USAGE

PRINT

PRINT permits a print file to be entered into the PRINT Queue from within a procedure.

General Format:

```
                      ⎧ ⎧filename ⎫ ⎡    ⎧libname  ⎫⎤ ⎡    ⎧volname  ⎫⎤⎫
[label:] ...   PRINT  ⎨ ⎩(refkey1)⎭ ⎢ IN ⎩(refkey2)⎭⎥ ⎢ ON ⎩(refkey3)⎭⎥⎬
                      ⎩  (spec-label)                                  ⎭

             [, CLASS = class]    ⎡, STATUS = ⎧SPOOL⎫ ⎤  [, FORM# = form]
                                  ⎣           ⎩HOLD ⎭ ⎦

             [, COPIES = copies]  ⎡, ⎧DISPOSITION⎫   ⎧SCRATCH⎫⎤
                                  ⎢  ⎩DISP       ⎭ = ⎨REQUEUE⎬⎥
                                  ⎣                  ⎩SAVE   ⎭⎦
```

Example:
Test: PRINT REPORT12 IN ABCREPS ON VOL444, CLASS=A, COPIES=2

PROCEDURE

PROCEDURE (PROC) defines a procedure; anything following the letters PROCEDURE or PROC on the same line is interpreted as a comment. PROCEDURE must be the first statement in a procedure.

General Format:

```
{ PROC      }
{ PROCEDURE }        [comment]
```

Example:
PROCEDURE   LOGON for user ABC

PROMPT

PROMPT displays text on the workstation screen and accepts data for variables. A feature of PROMPT is the ability to accept PF key values in a declared variable. With PROMPT, procedure execution is halted until the user responds appropriately to the prompt. PROMPT syntax is similar to MESSAGE.

General Format:

```
[label:] ... PROMPT  [PFKEY = variable]

{                          { string-constant  }   [              { string-constant  } ]     ]    }
{ [CENTER] [ [attribute]   { integer-constant }   [ , [attribute] { integer-constant } ] ... ; } ...
{                          { string-variable  }   [              { string-variable  } ]     ]    }
{                          { integer-variable }   [              { integer-variable } ]     ]    }

             [             { string-constant  }   [              { string-constant  } ]     ]
  [CENTER]   [ [attribute] { integer-constant }   [ , [attribute] { integer-constant } ] ...]
             [             { string-variable  }   [              { string-variable  } ]     ]
             [             { integer-variable }   [              { integer-variable } ]     ]
```

Example:
```
Test:  PROMPT PFKEY=&PF
                  "ENTER FILE = ", UPPER &FILE;;
       LINE "PRESS ENTER TO CONTINUE, PF16 TO EXIT"
```

PROTECT

PROTECT modifies file or library protection information; it is analogous to the PROTECT option of the Manage FILES/LIBRARIES command.

Format 1:

```
                        { {filename  }                                              }
[label:] ... PROTECT    { {(refkey1) }    [IN  {libname  }]   [ON  {volname  }]     }
                        {  (spec-label)        {(refkey2)}          {(refkey3)}     }

             TO  {[OWNER = owner]  [,PERIOD = period]  [,FILECLAS = fileclass]}
```

Example:
Test: PROTECT myfile IN mylib ON VOL2 TO FILECLAS=A

Format 2:

```
                                { {libname   }                      }
[label:] ... PROTECT LIBRARY    { {(refkey1) }    [ON  {volname  }] }
                                {  (spec-label)        {(refkey2)}  }

             TO  {[OWNER = owner]  [,PERIOD = period]  [,FILECLAS = fileclass]}
```

Example:
Test: PROTECT SALARIES TO OWNER=GSS, FILECLAS=Q

RENAME

RENAME allows the user to retitle a file or library; it is analogous to the RENAME option of the Manage FILES/LIBRARIES command.

Format 1:

$$\text{[label:] ... RENAME} \left\{ \begin{matrix} \left\{ \begin{matrix} \text{filename1} \\ \text{(refkey1)} \end{matrix} \right\} \\ \text{(spec-label)} \end{matrix} \right. \left[\text{IN} \left\{ \begin{matrix} \text{libname1} \\ \text{(refkey2)} \end{matrix} \right\} \right] \left[\text{ON} \left\{ \begin{matrix} \text{volname} \\ \text{(refkey3)} \end{matrix} \right\} \right] \right\}$$

$$\text{TO} \left\{ \begin{matrix} \text{filename2} \\ \text{(refkey4)} \end{matrix} \right\} \left[\text{IN} \left\{ \begin{matrix} \text{libname2} \\ \text{(refkey5)} \end{matrix} \right\} \right]$$

Example:
Test: RENAME ABCFIL IN deflib ON VOL444 TO xyzfil IN QRSLIB

Format 2:

$$\text{[label:] ... RENAME LIBRARY} \left\{ \begin{matrix} \left\{ \begin{matrix} \text{libname1} \\ \text{(refkey1)} \end{matrix} \right\} \\ \text{(spec-label)} \end{matrix} \right. \left[\text{ON} \left\{ \begin{matrix} \text{volname} \\ \text{(refkey2)} \end{matrix} \right\} \right] \right\}$$

$$\text{TO} \left\{ \begin{matrix} \text{libname2} \\ \text{(refkey3)} \end{matrix} \right\}$$

Example:
Test: RENAME USERLIB ON VOL444 TO ABCLIB ON VOLDEF

RETURN

RETURN unconditionally terminates procedure execution. It can be used either as a separate statement or as a clause in an IF statement.

General Format:

```
                            ┌ integer-variable   ┐   ┌     ┌ integer-variable ┐┐   ┐
[label:] ...  RETURN [ CODE = ┤ integer-constant   ├   [  +  ┤ integer-constant ├]...]
                            └ step-label         ┘   └     └ step-label       ┘┘   ┘
```

Example:
Test: RETURN CODE = LABEL1 + LABEL2 + 1000

RUN

RUN executes a program or procedure and is analogous to the RUN command on the Command Processor except that with RUN the user can pass parameters to the program or procedure being run.

General Format:

```
                  ┌┌ filename  ┐                                              ┐
[label:] ... RUN  ┤┤ (refkey1) ├  [ IN  { libname  } ]  [ ON  { volname  } ]  ├
                  └  (spec-label)        { (refkey2) }          { (refkey3) }  ┘

        [ USING  { variable }   [ ,  { variable } ] ... ]
                 { constant }        { constant }
```

Example:
Test: RUN DATE IN USERAIDS ON SYSTEM USING "HD", &date

SCRATCH

SCRATCH deletes a file or library from a specified volume. It is analogous to the SCRATCH option of the Manage FILES/LIBRARIES command.

Format 1:

$$\text{[label:] ... SCRATCH} \left\{ \begin{matrix} \left\{ \begin{matrix} \text{filename} \\ \text{(refkey1)} \end{matrix} \right\} \\ \text{(spec-label)} \end{matrix} \quad \left[\text{IN} \left\{ \begin{matrix} \text{libname} \\ \text{(refkey2)} \end{matrix} \right\} \right] \quad \left[\text{ON} \left\{ \begin{matrix} \text{volname} \\ \text{(refkey3)} \end{matrix} \right\} \right] \right\}$$

Example:
Test: SCRATCH USERFILE IN ABCLIB ON VOL444

Format 2:

$$\text{[label:] ... SCRATCH LIBRARY} \left\{ \begin{matrix} \left\{ \begin{matrix} \text{libname} \\ \text{(refkey1)} \end{matrix} \right\} \\ \text{(spec-label)} \end{matrix} \quad \left[\text{ON} \left\{ \begin{matrix} \text{volname} \\ \text{(refkey2)} \end{matrix} \right\} \right] \right\}$$

Example:
Test: SCRATCH LIBRARY USERLIB ON VOL444

SET

SET can specify library and volume default names, default file classes, print mode, and job submittal options. SET is analogous to the SET Usage Constants command on the Command Processor. Legal setkey keywords that identify fields for specifying default parameter values are as follows:

FILECLAS	INLIB	PRNTMODE
FORM#	INVOL	PROGLIB
JOBCLASS	OUTLIB	PROGVOL
JOBLIMIT	OUTVOL	PRTCLAS
JOBQUEUE	RUNLIB	PRTFCLAS
LINES	RUNVOL	SPOOLIB
PRINTER	WORKVOL	SPOOLVOL

General Format:

```
[label:] ...  SET { setkey1 = value1  [,setkey2 = value2]  ... }
```

Example:
Test: SET PROGLIB=USERPROG, PROGVOL=VOL444, FILECLAS=A

SUBMIT

SUBMIT places a procedure into the PROCEDURE Queue for noninteractive execution. It is analogous to the SUBMIT command on the Command Processor.

General Format:

```
                        { { procname   }                                           }
[label:]  ...  SUBMIT   { { (refkey1)  }  [ IN  { libname  } ] [ ON  { volname  } ] }
                        {   (spec-label)         { (refkey2) }         { (refkey3) }   }

              [AS    procedure-id]   [, CLASS = class]   [ , STATUS = { RUN  } ]
                                                                      { HOLD }

         [ DUMP = { PROGRAM } ] [ , CPULIMIT = { hh:mm:ss } ]
         [        { YES     } ] [              { ss       } ]
         [        { NO      } ]

       [ , ACTION = { CANCEL } ]   [ , { DISPOSITION } = REQUEUE ]
       [            { WARN   } ]   [   { DISP        }           ]
       [            { PAUSE  } ]
```

Example:
Test: SUBMIT JOB123 AS PAYROLL, CLASS=A, STATUS=HOLD,
CPULIMIT=0:1:30, ACTION=WARN

USING

USING declares the formal parameters to a procedure. It is optional. If the USING statement is present, it must immediately follow the PROCEDURE or PROC statement.

General Format:

```
USING  variable [, variable]  ...  AS  { STRING   (n) }
                                       { INTEGER      }

    [ , variable [, variable]  ...  AS  { STRING   (n) } ]  ...
    [                                   { INTEGER      } ]
```

Example:
USING &PARM1, &PARM2 INTEGER, &PARM3 STRING (8)

* The value (n) is an integer between 1 and 256, inclusive.

GLOSSARY OF TERMS

Certain parameters and ranges of terms are common to all Procedure language statements and are provided below for reference purposes.

TERM	DEFINITION
comment	Any user-written message. The Procedure Interpreter interprets comments as blanks.
fileclass	A one character value from among A-Z, #, $, ƀ, or @. Fileclass can also be a string-constant, string-variable, or substring.
filename	An alphanumeric value of up to eight characters that must begin with an alphabetic character, integer, @, $, or # and contain no embedded spaces. Can also be a string-constant, string-variable, or substring.
integer-constant	A sequence of one or more digits whose value is in the range -99999999 to 99999999.
integer-variable	Fullword signed integers whose value is in the range of -2147483648 to 2147483647.
label	An alphanumeric value of up to eight characters that must begin with an alphabetic character, contain no embedded spaces, and be followed by a colon (:), except when a step-label is referenced in an IF, GOTO, or ASSIGN statement, or used in a refkey. Labels identify procedure statements and are named according to the function being performed. Labels of statements which provide return codes (MOUNT, DISMOUNT, SCRATCH, RENAME, PROTECT, PRINT, SUBMIT, RUN) are termed step-labels; labels of parameter supplying statements (DISPLAY, ENTER) are termed spec-labels; labels of other statements (IF, GOTO, RETURN, EXTRACT, MESSAGE, PROMPT, DECLARE, ASSIGN, SET, LOGOFF) are termed stmt-labels.
libname	An alphanumeric value up to eight characters that must begin with either an alphabetic character, @, $, or # and contain no embedded spaces. Libname can also be a string-constant, string-variable, or substring.

TERM	DEFINITION
owner	A one to three character alphanumeric value that must begin with an alphabetic character and contain no embedded spaces. Owner can also be a string-constant, string-variable, or substring.
period	A numeric value in the range 0-999. Period can also be a constant, variable, or substring.
pfkey	A numeric value in the range 1-32. Pfkey can also be a constant, variable, or substring.
prname	Used in the DISPLAY or ENTER statement to identify the parameters specified in that statement.
refkey	Keyword identifying a field in a labeled DISPLAY or ENTER statement preceding the current statement. The value associated with this keyword in the referenced statement is obtained for the current field through backward reference. A refkey consists of the label of the referenced statement, followed by a period and the keyword identifying the referenced field. Refkeys must be enclosed in parentheses.
spec-label	The label of a previous specification statement (ENTER or DISPLAY) from which parameters are to be obtained through backward reference for use in the current statement.
step-label	Label of a DISMOUNT, MOUNT, PRINT, RUN, SCRATCH, SUBMIT, SET, RENAME, ASSIGN, or PROTECT statement. Used in IF and RETURN statements to identify the return code to be tested.
stmt-label	The label of a MESSAGE, PROMPT, EXTRACT, DECLARE, ASSIGN, SET, LOGOFF, IF, GOTO, or RETURN procedure statement.
string-constant	A text string up to 256 characters in length enclosed in single (') or double (") paired quotes.

TERM	DEFINITION
substring	A portion of a string-variable represented by the character position defined by "start" for a specified length. Substrings are allowed wherever a string-variable is allowed, except for the following statements: DECLARE, USING, RUN . . . USING (as a parameter).
unit#	A numeric value in the range 1-099. Unit can also be a string-constant, string-variable, or substring.
variable	Variable names can be 2 to 31 characters. The first character must be an ampersand (&); the other characters can be chosen from the characters A-Z, a-z, 0-9, @, $, #, and _. The content of a variable operand is the value of that operand. Variables can be either uppercase or lowercase. Lowercase characters are converted internally to uppercase.
verb	A command to carry out a specific task. Each statement in a procedure must begin with a Procedure language verb describing the operation to be performed.
volname	A one to six alphanumeric value that must begin with either an alphabetic character, integer, @, $, or # and contain no embedded spaces. Volname can also be a string-constant, string-variable, or substring.

PROCEDURE LANGUAGE RETURN CODE VALUES

Statement Name	Return Code	Meaning
DISMOUNT	4	Input volume name is blank, or bytes 0-1 in the input are nonzero.
	8	Volume cannot be found.
	12	Volume cannot be dismounted.
	16	Device detached.
	20	Volume in use by a user or the operating system.
	24	Volume reserved by another user.
	28	GETMEM (SVC) pool failure.
	32	Device is reserved by another task.
MOUNT	4	Successful mount, but new volume label type does not agree with the input parameters.
	8	Successful mount, but the new volume name is not the volume name requested.
	12	Disk or tape I/O error detected while reading the new volume label, or the new volume has a bad VTOC. VCBSER is set to blank.
	16	Device not a disk or tape, or invalid device number.
	20	Device detached.
	24	Volume type (REM or FIX) not found.
	28	Request to mount unlabeled volume on disk other than diskette.
	32	Input volume name is blank.
	36	Volume already mounted. Also set for a duplicate volume name.
	40	Volume is currently in use.
	44	Volume reserved by another user for exclusive use.
	48	I/O buffer is insufficient to perform the mount.
	52	Unable to allocate space for tape I/O control blocks.
	56	Invalid request: work and/or spool filing requested in a nonlabeled volume.
	60	Invalid request: nonstandard addressing attempted with standard label option or on a hard-sectored device.

PROCEDURE LANGUAGE RETURN CODE VALUES

Statement Name	Return Code	Meaning
MOUNT	64	Wrong media: soft-sectored diskette inserted into a device for hard-sectored diskettes only.
	68	Wrong media: hard-sectored diskette inserted into a device for soft-sectored diskettes only.
	72	Wrong media: hard-sectored diskette inserted for a nonstandard addressing request.
	76	Wrong addressing mode: caller requested MOUNT for standard addressing but diskette is nonstandard addressing.
	80	Device reserved by another user.
	84	PF16 was entered when the MOUNT message was displayed.
PROTECT	4	Volume not mounted.
	8	Volume used exclusively by other user.
	12	All buffers in use, no protection change.
	16	Library not found.
	20	File not found.
	24	Update access denied, no protection change.
	32	File in use, no protection change.
	36	VTOC error.
	40	VTOC error.
	44	Invalid argument list address.
	48	I/O error; VTOC unreliable.
	52	Open or protected files bypassed in protecting library.
	56	Invalid new protection data.
RENAME	4	Volume not mounted.
	8	Volume used exclusively by other user.
	12	All buffers in use, no rename.
	16	Library not found.
	20	File not found.
	24	Update access to file protection class denied, no rename.
	28	Unexpired file, no rename.
	32	File in use, no rename.
	36	VTOC error.
	40	VTOC error.
	44	Invalid argument list address.

Statement Name	Return Code	Meaning
RENAME	48	I/O error, VTOC unreliable.
	52	New filename or library name already exists, no rename.
	56	New filename invalid (or first character #), no rename.
	60	VTOC is currently full.
	64	Reserved bits in the parameter list options byte are nonzero.
SCRATCH	4	Volume not mounted.
	8	Volume used exclusively by other user.
	12	All buffers in use, no scratch.
	16	Library not found.
	20	File not found.
	24	Update access to file protection class denied (single-file scratch only).
	28	Unexpired file, no scratch (single-file scratch only).
	32	File in use, no scratch.
	36	VTOC error.
	40	VTOC error.
	44	Invalid argument list address.
	48	I/O error, VTOC unreliable.
	52	Open, protected, and/or unexpired file(s) bypassed in scratching library.
SUBMIT	4	Volume not mounted.
	8	Volume in exclusive use.
	12	All buffers in use; unable to perform verification.
	16	Library not found.
	20	File not found.
	24	Improper file type.
	28	File access denied.
	32	VTOC error.
	36	VTOC error.

WANG

Title VS PROCEDURE LANGUAGE QUICK REFERENCE

Publications Number 800-6201PP-03

Customer Comment Form

Help Us Help You

We've worked hard to make this document useful, readable, and technically accurate. Did we succeed? Only you can tell us! Your comments and suggestions will help us improve our technical communications. Please take a few minutes to let us know how you feel.

Please rate the quality of this publication in each of the following areas.

	VERY GOOD	GOOD	FAIR	POOR	VERY POOR
Technical Accuracy — Does the system work the way manual says it does?	☐	☐	☐	☐	☐
Readability — Is the manual easy to read and understand?	☐	☐	☐	☐	☐
Clarity — Are the instructions easy to follow?	☐	☐	☐	☐	☐
Examples — Were they helpful, realistic? Where there enough of them?	☐	☐	☐	☐	☐
Organization — Was it logical? Was it easy to find what you needed to know?	☐	☐	☐	☐	☐
Illustrations — Were they clear and useful?	☐	☐	☐	☐	☐
Physical Attractiveness — What did you think of the printing, binding, etc?	☐	☐	☐	☐	☐

What errors or faults did you find in the manual? (Please include page numbers) ____________

Do you have any other comments or suggestions? ____________

Name ____________

Company ____________

Street ____________

City ____________

State/Country ____________

Zip Code ____________ **Telephone** ____________

Thank you for your help.

WANG

Fold

Printed in U.S.A. 14-3151 8-82-5C

Cut along dotted line.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

WANG LABORATORIES, INC.
CHARLES T. PEERS, JR., MAIL STOP 1363
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Attention: Technical Writing Department

WANG

ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851
TEL. (617) 459-5000
TWX 710-343-6769, TELEX 94-7421

Printed in U.S.A.
800-6201PP-03
8-82-5M