SH20-1922-0

Installed User Program

Virtual Machine /370
Conversational Monitor System (VM/CMS)
EXEC Language Extensions
Program Description/Operations Manual

Program Number: 5796-PJA

This manual describes the functions of the processor. Formats and examples of the commands are included in the form of a user's guide. General system design and installation procedures are also specified.

PROGRAMMING SERVICES PERIOD

During a specified number of months immediately following initial availability of each licensed program designated as the PROGRAMMING SERVICES PERIOD, the customer may submit documentation to a designated IBM location when he encounters a problem which his diagnosis indicates is caused by a licensed program error. During this period only, IBM through the program sponsor(s), will without additional charge respond to an error in the current unaltered release of the licensed program by issuing known error correction information to the customer reporting the problem and/or issuing corrected or notice of availability of corrected code. However, IBM does not guarantee service results or represent or warrant that all errors will be corrected. Any onsite programming services or assistance will be provided at a charge.

WARRANTY

EACH LICENSED PROGRAM IS DISTRIBUTED ON AN 'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND EITHER EXPRESS OR IMPLIED.

First Edition (January 1977)

A form for readers' comments has been provided at the back of this publication. If this form has been removed, address comments to: IBM Corporation, B/O #115, 101 North Shoreline Drive, Corpus Christi, Texas 78401. Attention: Mr. Richard Brandle.

TABLE OF CONTENTS

INTRODUCTION 1

SYSTEM OVERVIEW. 1
  Description. 1
  System Flowchart 3
  Programming Systems. 4

INSTALLATION PROCEDURE 5
  Dumping the distribution tape. 5
  Assembling the EXSERV source 6
  Generating the EXSERV command module 7
  Verifying the installation results 7
  Use of Sample Execs. 7
  Release Restrictions 8
  Release Compatibility Considerations 8

EXSERV USER'S GUIDE. 10
  Notational Conventions 10
  Command Syntax 12
  Processor Interface. 12
  Interface Between EXSERV and CMS EXEC Interpreter. . . 13
  Return Codes 13

EXSERV FUNCTIONS 15
  BLDL (Performs OS BLDL function) 15
  CALC (Calculate function). 16
  DATETIME (Date and time services). 17
  DEVTYPE (Device type information). 18
  FIND (Search file for character string). 20
  GETOPTVL (Get option value). 21
  GETPARM/GETOPT (Get parameter or option) 23
  LOOKUP (Search a list) 25
  PRINT/PUNCH/STACK/TYPE (Handles untokenized output). . 26
  READ (Read CMS files). 28
  RETURN (Pass a value back) 30
  STATE/FILEDATA (Get file characteristics). 31
  STKFILE (Put a CMS file on console stack). 33
  SYSDATA (Obtain user or system information). 34
  SYSGLBL (Global variable processor). 35
  TSTOPT (Test option list). 37
  WRITE (Create or update a CMS file). 39

INTRODUCTION

This document is a guide for installing and understanding the functions of the CMS EXEC language extensions. As the command name for the extensions module is 'EXSERV', the extensions will be refered to as such throughout this document. Since EXSERV is an extension of the CMS EXEC processor, this manual assumes that the reader has a good working knowledge of CMS and the EXEC language.

The EXSERV module is designed to run in the transient program area of a CMS virtual machine. Therefore, it can be invoked by any CMS user in an EXEC procedure or EDIT macro. The user's guide section of this manual contains detailed descriptions and examples of the EXSERV commands. This may be extracted and included with a CMS user's guide.

Detailed instructions for the installation of the EXSERV module by a system programmer are included. It is assumed that the installer has a good understanding of CMS and the EXEC processors.

For details on the internal design of the command processors, refer to the EXSERV Systems Guide, Form. No. LY20-2279.

SYSTEM OVERVIEW

DESCRIPTION

The CMS exec language extensions are designed to provide additional functions to, and expand the usefulness of the Virtural Machine/370 CMS exec language. It is a standard CMS command module which runs in the transient area and allows the EXEC writer to:

1. Perform multiplication, division, and residue operations as well as addition and subtraction. ("CALC" function)

2. Obtain current date and time in EXEC variables. ("DATETIME" function)

3. Obtain real or virtual device type information. ("DEVTYPE" function)

4. Search a file in a specific column for a particular character string. ("FIND" function)

5. Search a list of values for a particular value. ("LOOKUP" function)

6. Determine if a member is defined in a library and obtain the starting record number of the member in the library and the size of the member. ("BLDL" function)

7. Obtain the values of specific parameters or options from the EXEC argument list and supply default values if no value was provided. ("GETOPT/GETPARM" function)

8. Obtain the value of an option which follows a specific keyword and supply a default if the keyword does not occur in the option list. Also, several synonyms may be provided for the keyword if desired. ("GETOPTVL" function)

9. Return the values of local EXEC variables to the next higher level EXEC. ("RETURN" function)

10. Print, punch, stack, or type data which contains a specific number of embedded blanks and strings greater than 8 characters in length. ("PRINT/PUNCH/STACK/TYPE" function)

11. Read any record in a CMS file into EXEC variables. ("READ" function)

12. Place a CMS file in the console stack. ("STKFILE" function)

13. Obtain data from the File System Table (FST) when only part of the file ID is known. ("FILEDATA/STATE" function)

14. Obtain virtual machine user ID, system level number, or virtural machine storage size. ("SYSDATA" function)

15. Set global EXEC variables which may be accessed from command to command. ("SYSGLBL" function)

16. Test the option list for one of a mutually exclusive set of values and return the value found or a default value if none were found. Also, short forms of the option values may be specified if desired. ("TSTOPT" function)

17. Add new records to a CMS file, rewrite any record in a CMS file, or overlay specific columns of a record in a CMS file. ("WRITE/WRTUPD" function)

Items 5, 7, 8, and 16 greatly ease the task of obtaining and testing values of parameters and options passed in an EXEC argument list.

Items 4, 6, 11, 12, and 17 provide I/O capability for the EXEC writer.

Item 10 provides a way to print, punch, stack, or type data with variable substitution and/or character strings longer than eight characters.

SYSTEM FLOWCHART

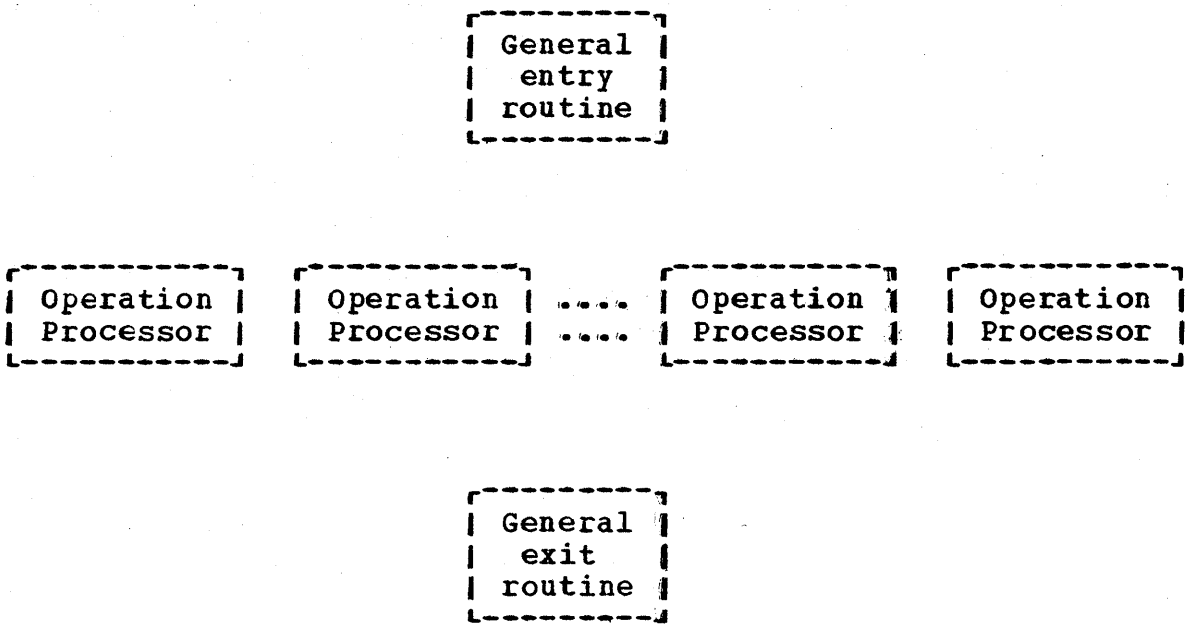
EXSERV consists of:

* a general entry routine to set up the work areas, check the validity of the parameter list, and branch to the correct processor

* 17 operation processors to perform the functions

* 11 subroutines that are common to several of the processors

* a general exit routine that sets the return code and restores the original environment.



PROGRAMMING SYSTEMS

EXSERV is written in CMS Assembler and is designed to run under VM/370 - Release 3.0. For operation in a previous release, some of the functions may not be supported. Consult the Installation section of this document and the function in question.

Minimum hardware configuration is the same as for VM/370.

The installation procedure for EXSERV is divided into four steps. Specifically:

1) Load distribution tape onto a CMS minidisk.
2) Assemble the EXSERV source.
3) Generate the EXSERV command module.
4) Execute the basic installtion verification procedure.

Each of these steps will be described in detail.

DUMPING THE DISTRIBUTION TAPE

The distribution tape was created using the CMS 'TAPE DUMP' command and must be restored with the CMS 'TAPE LOAD' command. Due to dependencies in the EXSERV generation execs, the minidisk must be accessed as an A-DISK.

The tape is divided into two files. The first file contains the necessary material to generate EXSERV and the second file contains functional sample execs which use EXSERV. Additionally, a technique for an exec testing facility is provided. If the second file is not desired, omit the '(EOT)' specification on the 'TAPE LOAD' command.

The CMS minidisk space requirements for the distribution material and generation procedure are as follows:

Distribution material	340 blocks
Generation procedure output	650 to 970 blocks
Assembler workspace	1300 blocks

On a 3330 drive, this is approximately 10 cylinders.

The only output required for operation of EXSERV is the module file. This is the actual EXSERV command module and requires approximately 8 blocks on a formatted CMS minidisk. All other files may be erased. However, all maintenance changes will be provided in source form.

With the distribution tape loaded and attached as device address 181, perform the following set of commands:

```
TAPE REW                          Operator input
R;                                System output
TAPE LOAD * * A (EOT)             Operator input
LOADING ...                       System output
EXSERV   ASSEMBLE A0
EXSRVASM EXEC     A0
EXSRVGEN EXEC     A0
```

```
EXSRVVFP EXEC     A0
EXSPL100 UPDATE   A0
EXSPL217 UPDATE   A0
END-OF-FILE OR END-OF-TAPE
$EXENTER EXEC     A0
EXHT     EXEC     A0
EXRT     EXEC     A0
EXTEST   EXEC     A0
FILES    EXEC     A0
HELP     EXEC     A0
END-OF-FILE OR END-OF-TAPE
END-OF-FILE OR END-OF-TAPE
R;
```

This concludes the dumping of the distribution material.

ASSEMBLING THE EXSERV SOURCE

The exec 'EXSRVASM' is used in the next step to assemble the source of EXSERV. This exec has one parameter which indicates the release level of VM/370 in which EXSERV will operate. This parameter may have the following values:

- 100 - Indicates EXSERV is to be assembled for use on a VM/370 system prior to Release 2 PLC 17.
- 217 - Indicates EXSERV is to be assembled for use on a VM/370 system between Release 2 PLC 17 but prior to Release 3
- blank - Indicates EXSERV is to be assembled for use on Release 3 of VM/370.

If other than a Release 3 VM/370 system is the host, a temporary source module (named $EXSERV) will be created for the assembly step. The original source module is not modified. The changes being incorporated are one to two card changes which reflect a change in the starting displacement of the EXEC Interpreter common area. If problems arise after a release change, this is the most likely cause of the problem. Refer to CMS module 'DMSEXT' for a mapping of this area.

During the course of the source update and assembly, status messages will be issued. At the conclusion of this exec, print the 'EXSERV LISTING' file for future reference.

GENERATING THE EXSERV COMMAND MODULE

Upon completion of the EXSERV assembly, the exec 'EXSRVGEN' must be run to generate the command module for EXSERV. This exec has no parameters. The output of this exec will be the EXSERV command module file and the load map. When this exec completes, EXSERV is ready to use. If desired, the EXSERV module file may be copied to the CMS system disk. Be sure to save a new CMS system or else the file will not be in the saved system's file directory for the S-DISK.

VERIFYING THE INSTALLATION RESULTS

After the installation process is complete, a basic verification procedure ('EXSRVVFP') may be executed to insure EXSERV is working correctly. This exec will use four EXSERV functions to insure that variables may be passed correctly between EXSERV and the EXEC interpreter. The four functions used are:

DATETIME
DEVTYPE
SYSDATA
TYPE

These functions were selected because they do not require user input or user files for operation. As the exec executes, messages will be issued which indicate the expected results. At no time should a nonzero return code or program check result. If the verification procedure fails, insure the correct release level was specified on the 'EXSRVASM' exec command for the assembly of EXSERV.

USE OF SAMPLE EXECS

The sample execs in the second file may be used for additional testing and contain examples of how EXSERV can be used. Clearly, without EXSERV these functions could not be performed in an exec without the use of external modules written in assembly language.

Execs 'EXTEST', 'FILES', and 'HELP' utilize EXSERV in their operation. Execs 'EXHT' and 'EXRT' are used internally by the 'FILES' exec and are part of an exec testing facility. The format, usage, and parameters required for any of these execs may be obtained by entering 'HELP exec-name'.

The exec named '$EXENTER' is an edit macro and should not be

used from the console. Its used is explained in the help information of the 'EXTEST' exec.

The 'HELP' exec is a generalized function which can be useful in an operational environment. HELP information is written at the end of an exec in the following format:

(A full line of asterisks)
FORMAT:
    Exec format information
(Blank line)
USAGE:
    Description of exec function and usage.
(Blank line)
PARMS:
    Detailed description of parameter and option values.

The operation of the HELP exec depends on this format. Analysis of the exec logic will reveal the reasons.

RELEASE RESTRICTIONS

SYSDATA Function:

This function uses the VM/370 diagnose code zero to extract the user id and system level number. The diagnose code zero function was not implemented until release 2 PLC 19 of VM/370. If SYSDATA is used on a prior release, the return code will always be four.

Also, if the virtural machine storage size is requested, the VM/370 diagnose code sixty function is used. This function was not available until release 3 PLC 1 of VM/370. Therefore, if the machine storage size is requested on a system between release 2 PLC 19 and release 3 PLC 1, the results will be unpredictable.

RELEASE COMPATIBILITY CONSIDERATIONS

EXSERV in general:

EXSERV interfaces to the exec interpreter thru the interpreter work area defined in CMS module DMSEXT. The mapping of this area is available only within this module. A second independent mapping is also contained in the EXSERV source. When changing releases, it is important to determine if any updates have been applied to DMSEXT which affect the displacements of fields in the interpreter work area. If so, duplicate changes must be made in the EXSERV source. The interpreter work

area format is moderately stable and has changed twice during the lifetime of VM/370. These changes were required for new functions added to either VM/370 or the exec interpreter.

SYSGLBL Function:

This function uses one reserved word in the low storage area of the CMS nucleus. (Area 'NUCRSV1' mapped by the dsect NUCON) This word is used to anchor the global variable chain and has remained unused thru the current release of VM/370. Future releases may utilize this area for new functions in which case a different reserved area must be selected. During the implementation of a new release, examine the NUCON macro for possible changes regarding this area.

EXSERV USER'S GUIDE

This guide shows the proper syntax, function, examples, and possible uses of the EXSERV operations.

NOTATIONAL CONVENTIONS

The notation defining the command syntax in this guide is described in the following paragraphs:

1. Truncations and abbreviations of commands

   Where abbreviation of a command is permitted, uppercase letters represent the shortest possible version of the command. The example below shows the format specification for the EXEC command.

   ```
   EXec
   ```

   This representation means that EX, EXE, and EXEC are all valid specifications for this command name.

   Options are specified in the same manner. If no minimum truncation is noted, you must enter the entire word.

   A second series of characters below the command name or option indicates that an abbreviation (a collection of characters which is not a simple truncation) is also valid.

   ```
   MESSAGE  Full command name (MESSAGE)
   MSG      Valid abbreviation (MSG)
   ```

2. The following symbols define the command format and should never be typed when you enter the actual command.

underscore	_
braces	{ }
brackets	[ ]
ellipsis	...

3. You should type uppercase letters and words (Note: The Editor automatically converts lowercase letters into uppercase before the file is written) and the symbols listed below, as specified in the statement format.

asterisk	*
comma	,
percent sign	%
equal sign	=
parenthesis	( )
period	.
colon	:

4. Lowercase letters, words, and symbols that appear in the command format represent variables for which you should substitute specific information. For example, "fn ft fm" indicates that you should type file identifiers such as "MYFILE EXEC A1".

   Lowercase letters containing an imbedded hyphen (-) represent EXEC variables which should be entered less their leading ampersand (&). For example, "var-v1" indicates that you should type identifiers such as "VAR", which represents the EXEC variable "&VAR".

5. Choices are represented in the command format by stacking.

   A
   B
   C

6. An underscore indicates an assumed default option. If you select an underscored choice, you need not type it when you enter the command.

   EXAMPLE
   The representation:

   A
   B
   C

   indicates you may select either A, B, or C. However, if you select B, you need not type it since it becomes the default choice.

7. The use of braces denotes choices, one of which you must select.

   EXAMPLE
   The representation:

   { A }
   { B }
   { C }

   indicates you must specify either A, B, or C. If neither brackets nor braces enclose a list of choices, treat it as if enclosed by braces.

8. The use of brackets denotes choices, one of which you may select.

   EXAMPLE
   The representation:

```
[ A ]
[ B ]
[ C ]
```

   indicates that you may code A, B, or C, or you may omit the field.

9. An ellipsis indicates that you may repeat the preceding item or group of items more than once in succession.

   EXAMPLE
   The representation:

```
(option ...)
```

   indicates that you may code more than one option within the parenthesis.

COMMAND SYNTAX

The argument list to the EXEC procedure using EXSERV is divided into the following fields:

```
[EXEC] execname [parameters] [ ( options [ ) ] ]
```

"parameters" are any tokens between the EXEC name and the first left parenthesis. "Options" are any tokens between the first left parenthesis and the end of the EXEC argument list. An optional ending right parenthesis is not part of the option list.

PROCESSOR INTERFACE

The EXSERV Processor is always invoked by a command in the following format:

```
EXSERV operation [parameters] [ ( options [ ) ]]
```

Not all operations require parameters or option lists. Because EXSERV runs in the transient area, it may be used in "EDIT" macros.

INTERFACE BETWEEN EXSERV AND CMS EXEC INTERPRETER

1) Upon entry, EXSERV obtains the value of the CMS EXEC Interpreter's register 13 from the save area. This register points to the EXEC Interpreter's common area and allows EXSERV to access the local variables currently in use by the EXEC procedure. Also, additional variables may be created and inserted into the EXEC procedure's variable list.

2) If the global EXEC variables are used, one reserved word is used in the CMS nucleus ("NUCRSV1"). This function can be release sensitive.

3) The CMS User's Guide (GC20-1819) states that a percent sign (%) may be used as a place holder. The use of the percent sign as a place holder only applies to commands which are processed by the EXEC interpreter. The CMS command processor does not place any special significance on the percent sign character and treats it like any other character.

   EXSERV uses the percent sign frequently as a place holder. However, the EXEC writer is advised that different EXEC control words treat the percent sign differently. At times it may be necessary to change an argument returned by EXSERV from a percent sign to a null (blank) before using it.

4) EXSERV makes a very important distinction between variables and values that are passed to it. Any variable names that are passed to EXSERV must not be preceded by their ampersand (&). This is because the EXEC Interpreter changes any name preceded by an ampersand into its corresponding value before executing the command. Therefore, when a command asks for a variable name, it must be typed without the ampersand (&). If a value is needed, then it can be in the form of a literal or a variable name with the ampersand.

RETURN CODES

```
  0 - *****
  4 - *****     DEPENDS ON THE EXSERV FUNCTION
  8 - *****
 12 - *****
100 - EXSERV not called from within an EXEC.
104 - Invalid EXSERV operation request.
108 - Invalid EXSERV parameter list.
200 - Invalid   variable   name.   Name   exceeds   seven
```

characters plus an ampersand.

300 - Invalid decimal value (contains non-decimal digits).

304 - Numeric value exceeds eight characters after conversion.

308 - Invalid arithmetic operator.

312 - Invalid range for numeric value. (eg. a value used for a card column is greater than 80)

316 - Invalid hexadecimal value (contains non-hex digits).

4xx - Error from "FSSTATE". xx is STATE return code.

5xx - Error from "FSREAD". xx is READ return code. "EOF" (return code 12) is not considered an error and is returned as '12'.

6xx - Error from "FSWRITE". xx is the WRITE return code.

NOTE: The error return codes for the three macros mentioned above can be found in the CMS Command and Macro Reference, form number GC20-1818.

BLDL (Performs OS BLDL function)

FUNCTION:

This routine performs an OS 'BLDL' operation on a macro or text library. It searches the library directory and returns the beginning location of the member in the library and the length of the member. These values might be used in a COPYFILE command to move the library member into a seperate file of its own.

COMMAND FORMAT:

EXSERV BLDL mbrnm fn ft [fm][(str-var [ln-var [)]]]

mbrnm	- Member name for search.
fn	- Library file name.
ft	- Library file type (Normally MACLIB or TXTLIB).
fm	- File mode.
str-var	- Variable name in which to place the starting location of the member in the library.
ln-var	- Variable name in which to place the length of the member.

RETURN CODES:

0 -- BLDL SUCCESSFUL.
4 -- BLDL NOT SUCCESSFUL.
8 -- DIRECTORY UPDATE IN PROGRESS.

```
&CONTROL OFF
EXSERV BLDL &1 &2 MACLIB A (START LNGTH)
&IF &RETCODE NE 0 &EXIT &RETCODE
COPYFILE &2 MACLIB A &1 MACRO A
         (FROM &START FOR &LNGTH)
EDIT &1 MACRO A
MACLIB REP &2 &1
ERASE &1 MACRO A
&EXIT &RETCODE
```

```
exec editmac  foo mymaclib
R;
EDIT:
. . .
. . .
```

A simple EXEC (EDITMAC) which easily edits members of macro libraries.

CALC (Calculate function)

FUNCTION:

This function provides the EXEC writer with the capability to perform one of five mathematical operations.

COMMAND FORMAT:

EXSERV CALC ans-var = var1 opr var2

- ans-var - Name of the variable result is to be placed in.
- var1 - First operand. May be variable name or constant value.
- opr - Operation to be performed. + (addition), - (subtraction), * (multiplication), / (division), or // (remainder from division).
- var2 - Second operand. May be variable name or constant.

RETURN CODES:

0 -- RESULT IS ZERO OR POSITIVE
4 -- RESULT IS NEGATIVE

For division, it is necessary to invoke CALC twice if both the quotient and remainder are desired.

```
&CONTROL OFF
EXSERV CALC ANSWER = &1 &2 &3
&TYPE IT IS &ANSWER
&EXIT &RETCODE
```

```
exec whatis 7 * 8
IT IS 56
R;

exec whatis 10 / -2
IT IS -5
R(00004);
```

A simple EXEC (WHATIS) using the CALC function.

DATETIME (Date and time services)

FUNCTION:

This function allows the user to obtain the date and time in printable form in EXEC variables.

COMMAND FORMAT:

```
EXSERV { DATETIME }
       { DT       }
```

RETURNED VARIABLES:

The variable &DATE, in the format mm/dd/yy, and &TIME, in the format hh:mm:ss, will be available for use upon return from this command.

RETURN CODES:

NONE

```
&CONTROL OFF
EXSERV DATETIME
&TYPE TODAY IS &DATE
&TYPE THE TIME IS NOW &TIME
&EXIT &RETCODE
```

```
exec date
TODAY IS 08/25/76
THE TIME IS NOW 13:27:07
R;
```

A simple EXEC (DATE) using the DATETIME function.

DEVTYPE (Device type information)

FUNCTION:

This routine is used to extract device type information about non-spooled virtual devices. Both virtual and real device type information is available if desired.

COMMAND FORMAT:

```
EXSERV { DEVTYPE } rtn-v1 [rtn-v2] [ ( devadr [) ]]
       { DVT     }
```

rtn-v1 - Return variable name for real device information.
rtn-v2 - Return variable name for virtual device information.
devadr - Virtual device address. If omitted, the virtual console will be assumed.

RETURN CODES:

0 -- VALID DEVICE ADDRESS. INFORMATION RETURNED.
4 -- INVALID DEVICE ADDRESS. VARIABLES SET TO '%'.

NOTES:

SPOOLED DEVICE INFORMATION CANNOT BE EXTRACTED. IF AN ATTEMPT IS MADE TO OBTAIN INFORMATION ABOUT SPOOLED DEVICES, EXSERV WILL EXIT WITH RETURN CODE OF 4.

DEVICE CODES:

The information returned consists of a four digit number which is the device class and type codes as defined in the VM/370: System Programmer's Guide (GC20-1807). The first two digits define the device class, which are as follows:

80 - Terminal device
40 - Graphics device
20 - Unit record input device
10 - Unit record output device
08 - Magnetic tape device
04 - Direct access storage device
02 - Special device

The last two digits define the device type, which depends on what class it is. The complete codes for some of the more common devices are:

8040 - 2700 Binary synchronous line
8018 - IBM 2741 Communication terminal
4004 - IBM 3277 Display station
1041 - IBM 1403 Printer
0410 - IBM 3330 Disk storage facility

0408 - IBM 3350 Disk storage facility

```
    . . .
    . . .
    EXSERV DEVTYPE CONTYPE
    &IF &CONTYPE NE 4004 &EXIT &RETCODE
    CP SET PF1 COPY
    CP SET PF2 IMMED QUERY TIME
    &EXIT &RETCODE
```

Using DEVTYPE in a PROFILE EXEC to set the program function keys on a 3270.

FIND (Search file for character string)

FUNCTION:

This routine can be used to search a CMS file for a character string starting in a particular column.

COMMAND FORMAT:

```
EXSERV FIND fn ft [ fm [ rcd ]]( [cc] str rcd-var [) ]
```

fn	- Name of file to be searched.
ft	- Type of file to be searched.
fm	- Mode of file tc be searched.
rcd	- Starting record number for search. The default record is one.
cc	- Column in which to search for string. Defaults to one if not provided.
str	- Character string to be searched for. May be 1 to 8 characters.
rcd-var	- variable name which is to contain the record number of the record in which the search string was found.

RETURN CODES:

0 -- ITEM FOUND. VARIABLE SET TO RECORD NUMBER.
4 -- ITEM NOT FOUND. VARIABLE NOT SET.

```
. . .
. . .
EXSERV FIND &1 &2 A 1 ( 1 // RECORD)
&IF &RETCODE EQ 0 &SKIP 2
&TYPE JCL CARD NOT FOUND
&EXIT &RETCODE
. . .
. . .
```

Part of an EXEC which searches a file for OS JCL records.

GETOPTVL (Get option value)

FUNCTION:

The GETOPTVL Processor is used to extract option values which are preceeded by a specific keyword. This function is useful when positional values are not desirable. Also, several synonyms may be provided for the keyword. For EXEC procedures which depend on several options, this can eliminate most of the searching for the values and their positional dependency.

COMMAND FORMAT:

```
EXSERV { GETOPTVL } [ndx] rtn-var [syn-nm ...][ (dftval [) ]]
       { GOV      }
```

- ndx - Index number of value to be obtained. Defaults to one. (First value after keyword)
- rtn-var - Keyword name and name of variable in which the value following the keyword is to be set.
- syn-nm - Alternate keyword name. If a synonym keyword is used, the 'rtn-var' name is still used to return the value following the synonym name.
- dftval - Default value to be used if the keyword or any synonyms is not found in the option list. If not provided, the 'rtn-var' will be set to '%'.

RETURN CODES:

0 -- SUCCESSFUL COMPLETION. KEYWORD SET TO VALUE.
4 -- NO VALUE PROVIDED AFTER KEYWORD. KEYWORD OR SYNONYM OCCURRED AS LAST VALUE IN THE OPTION LIST OR INDEX POINTS PAST END OF EXEC OPTION LIST.

EXAMPLE:

The following section of an EXEC procedure could be used to get a variable number of disk filemodes following the keyword 'DISK' or 'DISKS'. If the procedure was called 'FOO', it could be invoked as follows:

```
EXEC FOO ... ( ... DISK [fm [fm ...]] ... [) ]
```

```
    &N = 1
    -LOOP
    EXSERV GETOPTVL &N DISK DISKS (%)
    &IF &RETCODE EQ 4 &GOTO -END
    &IF &DISK EQ % &GOTO -END
    * (statements that would verify that &DISK is valid.)
    * (for example, use EXSERV LOOKUP.)
    &DISK&N = &DISK
    &N = &N + 1
    &GOTO -LOOP
    -END
    &IF &N EQ 1 &GOTO -OUT
    . . .
    . . .
```

GETPARM/GETOPT (Get parameter or option)

FUNCTION:

The GETPARM/GETOPT Processor allows the EXEC coder to extract values from either the 'parameters' or 'options' which were input to the EXEC procedure. Everything up to the first left parenthesis (excluding the EXEC name) is considered a parameter. Everything after the first left parenthesis is considered an option. If a parameter or option value is not explicitly provided in the EXEC's argument list, a default value may be provided. Also, a specific parameter or option value may be obtained by providing its index number in the parameter or option list.

COMMAND FORMAT:

```
EXSERV { GETPARM } [ndx] rtn-var ... [ ( dftval ... [)]]
       { GP      }
       { GETOPT  }
       { GO      }
```

ndx - Index number of parameter or option value to be obtained. If not provided, the default is one greater than the index value used for the previous 'rtn-var' specified. If no such previous index value exists, it defaults to 1.

rtn-var - Name of the EXEC variable in which the parameter or option value is to be set. If the parameter or option was not included in the EXEC parameter list, the default value (dft-val) associated (positionally) with this variable will be used. If no default value was provided, the variable will be set to '%'.

dftval - Default value to be used if parameter or option value was not provided. Default values are associated on a one-to-one basis with return variables.

RETURNED VARIABLES:

The variable '&PNUM' or '&ONUM' is returned to the calling EXEC. if GETPARM is specified, &PNUM is returned, and for GETOPT, &ONUM is returned.
'&PNUM' provides the true number of parameters that are provided in the EXEC's argument list and '&ONUM', the true number of option values in the argument list.

EXAMPLE:

Assume an EXEC has been invoked with the following command line:

```
EXEC ANYNAME ABC DEF (GHI JKL)
```

And the following EXSERV command has been executed from within the EXEC:

```
EXSERV GETPARM FN FT FM (% SCRIPT A1)
```

Then upon return from EXSERV, the variables '&FN', '&FT', and '&FM' would be set to 'ABC', 'DEF', and 'A1'. 'ABC' and 'DEF' from the EXEC argument list, and 'A1' supplied by the default value. &PNUM would be set to 2. This supplies more information than &INDEX which would be set to 6. (Parenthesis are counted by &INDEX). The following EXSERV command would have the same results:

```
EXSERV GETPARM 2 FT 1 FN 3 FM (SCRIPT % A1)
```

LOOKUP (Search a list)

FUNCTION:

The function of this routine is to look for a value in a list of values and indicate if the value occurs in the list. This can be very useful in checking the validity of a variable.

COMMAND FORMAT:

```
EXSERV { LOOKUP } tstval ( tblvl ... [) ]
       { LU     }
```

tstval - Value to be searched for in a list of values.
tblvl - One or more values which comprise the list to be searched.

RETURN CODES:

0 -- VALUE LOCATED IN LIST.
4 -- VALUE NOT LOCATED IN LIST.

```
. . .
. . .
&TYPE WHICH DISK MODE ?
&READ &VARS &MODE
EXSERV LOOKUP &MODE (A B C D E F G S Y Z)
&IF &RETCODE EQ 0 &SKIP 2
&TYPE INVALID MODE - &MODE
&EXIT 4
. . .
. . .
```

Fragment of an EXEC procedure showing typical use of LOOKUP function.

PRINT/PUNCH/STACK/TYPE (Handles untokenized output)

FUNCTION:

This routine is used to print, punch, stack, or type data which contains strings greater than eight characters in length. The input tokens must be eight characters or less with blank characters indicated by an underscore character. Variables may be substituted into the data to be printed, punched, stacked, or typed.

If the print or punch function is used, a 'CP CLOSE' must be issued by the exec when all data has been printed or punched.

COMMAND FORMAT:

```
EXSERV { PRINT [ctl]   } ( [cc] data ...[)]
       { PRT [ctl]     }
       { PUNCH         }
       { PCH           }
       { STACK [order] }
       { STK   [order] }
       { TYPE          }
```

ctl - ASCII carriage control character.

order - Is either of the following. If neither is specified, FIFO is the default.

1) FIFO - Requests this line to be stacked at the end of the console stack.
2) LIFO - Requests the line to be placed at the beginning of the console stack.

cc - Card column in which data is to be printer, punched, stacked, or typed.

data - Data to be output. (Underscore characters are translated to blanks prior to output)
NOTE- Data starting with a numeric character is always considered to be a card column number.

RETURN CODES:

NONE.

EXAMPLE:

The data to be printed, punched, stacked, or typed should not start with a numeric value because it would

then be interpreted as a column number. Also, since CMS tokenizes all parenthesis, the token '(25,6)' would be received as '(25,6)', or as 3 tokens. The '25,6' would then be treated as a column number and an error would result. These problems can be overcome either by preceding numeric data with an underscore or inserting numeric data into a variable.
For instance:

```
&NUMBER = 24,2
EXSERV PUNCH (// 16 SPACE= (CYL,( NUMBER )))
```

would result in the following card:

```
col 1           col 16
|               |
//              SPACE=(CYL,(24,2))
```

All blanks between the data must be inserted explicitly since EXSERV concatenates all the data unless column numbers are specified. For instance, assume the variable '&UID' contains the value 'DOSMAINT'. Then the following command:

```
EXSERV PUNCH (//_JOB_ TEST_ 36 ***_JOB _FOR_ UID _***)
```

would produce the following card:

```
col 1                   col 36
|                       |
// JOB TEST             *** JOB FOR DOSMAINT ***
```

Variable names must not have underscores before or after them or they will not be substituted.

READ (Read CMS files)

FUNCTION:

This routine is used to read CMS files from within EXEC procedures. The data is returned in EXEC variables. Specific columns may be read if desired and blanks are not considered as part of the data. Character strings longer than eight characters are returned in successive variables.

COMMAND FORMAT:

```
EXSERV { READ } fn ft [fm [rcd [trcval]]] [ ([$COL] [cc] rtn-var ...[]) ]
       { RD   }
```

fn	- File name to be used for read.
ft	- File type to be used for read.
fm	- File mode to be used for read.
rcd	- Record number to read. Defaults to one.
trcval	- Last position in record to scan for data. The default is the record length if not specified.
$COL	- Requests the column number to be returned along with the data. A variable preceded by '&$' followed by the data variable name (rtn-var) will contain the column number. This option is only specified once at the beginning of the option list.
rtn-var	- Variable name in which data is returned.

RETURN CODES:

0 -- READ SUCCESSFUL.
12 -- END OF FILE.
5xx -- I/O ERROR DURING READ. 'xx' INDICATES RETURN CODE FROM 'FSREAD' MACRO.

EXAMPLE:

To read the fourth data item in the Nth record of 'MYFILE DATA', the following statement could be used:

```
EXSERV READ MYFILE DATA % &N ($COL % % % NUMBER)
```

After completion, the variable '&$NUMBER' would contain the starting column number of the data item, and &NUMBER would contain the data. Keep in mind that this processor 'tokenizes' the input record. That is, it breaks up any character strings longer than 8

characters into two or more strings. Therefore, in the above example, if the record contained a continuous string, then '&NUMBER' would contain the substring from columns 25 to 32.

RETURN (Pass a value back)

FUNCTION:

This routine is used to pass one or more local variable values to the next higher level EXEC. The next higher level EXEC can access the value of the variables by using the variable name within a command line. This provides EXEC procedures with a subroutine-like facility.

COMMAND FORMAT:

```
EXSERV { RETURN } var ...
       { RTN    }
```

var - Name of variable to be passed to the next higher level EXEC.

RETURN CODES:

0 -- VARIABLES SUCCESSFULLY PASSED TO NEXT HIGHER LEVEL.
4 -- NO HIGHER LEVEL EXEC TO PASS TO.

```
. . .
. . .
&X = &X + 1
&IF &X LT 100 &GOTO -LOOP
EXSERV RETURN X Y
&EXIT &RETCODE
```

Section of an EXEC procedure which does some calculations and returns two values.

STATE/FILEDATA (Get file characteristics)

FUNCTION:

The state processor can be used to determine if a file exists. Also, if a file does exist, the explicit file name, file type, and file mode can be obtained.

The original function of the 'STATE' operation is obsolete since the addition of the NOMSG option of the &CONTROL exec verb. However, the function remains since it is an integral part of the FILEDATA routine.

COMMAND FORMAT:

```
EXSERV { STATE    } fn [ft [fm]] [( rtn-fn [rtn-ft [rtn-fm]] [)]]
       { FILEDATA } fn [ft [fm]] [( rtn-fn [rtn-ft [rtn-fm [rtn-ff
       { FD       }              [rtn-lrl [rtn-ic [rtn-bc [rtn-dt
                                 [rtn-tm]]]]]]] [)]]
```

fn - File name for the search. May be specified as '*'.
ft - File type for the search. If omitted, defaults to '*'.
fm - File mode for the search. If omitted, defaults to '*'.

If the state operation is successful (return code 0), the following option values represent variable names in which the associated file information will be returned:

rtn-fn - Actual file name.
rtn-ft - Actual file type.
rtn-fm - Actual file mode.
rtn-ff - Actual file format ('F' or 'V').
rtn-lrl - Actual logical record length.
rtn-ic - Number of logical records in file.
rtn-bc - Number of physical 800-byte blocks.
rtn-dt - Date file created (mm/dd/yy).
rtn-tm - Time file created (hh:mm).

NOTE- If STATE/FILEDATA operation is unsuccessful, any return variables are set to '%'.

RETURN CODES:

0 -- FILE DOES EXIST. INFORMATION RETURNED.
28 -- FILE DOES NOT EXIST.
4xx -- 'xx' IS ANY RETURN CODE POSSIBLE FROM THE 'FSSTATE' MACRO. Return code 28 is not considered an error condition.

EXAMPLE:

A percent sign (%) may be used as a return variable name if a particular data item is not desired. For instance:

```
EXSERV FILEDATA NAME TYPE * (% % FM)
```

will extract the file mode of the file named 'NAME TYPE'.

STKFILE (Put a CMS file on console stack)

FUNCTION:

This routine is used to place all or part of a CMS file onto the console stack.

COMMAND FORMAT:

```
EXSERV { STKFILE }  fn ft [fm [strtrcd]][ {([[FOR] nbrrcds][) ]} ]
                                          {([TO rcdnbr][) ]    } ]
       { SF      }
```

fn - Name of file to be stacked.
ft - Type of file to be stacked.
fm - Mode of file to be stacked. Defaults to '*'.
strtrcd - Starting record number in file. Defaults to one.
nbrrcds - Number of records to be stacked.
rcdnbr - Last record number to be stacked.

NOTE - If neither 'nbrrcds' or 'rcdnbr' is specified, all records from the starting record number to the end of the file will be stacked.

EXAMPLE:

Suppose there existed a file called 'CHANGES EDIT' which contained several edit statements. If only a subset of these changes was desired, then the variables '&START' and '&STOP' would have to be initialized to the correct records. Then by executing the two following commands, any file can be easily edited:

```
EXSERV STKFILE CHANGES EDIT % &START (TO &STOP)
EDIT MYFILE SCRIPT
```

SYSDATA (Obtain user or system information)

FUNCTION:

The function of this routine is to return system ID variables to the user. The following information can be obtained:

1) USER ID
2) SYSTEM, MODIFICATION, AND PLC LEVEL NUMBERS.
3) VIRTURAL MACHINE STORAGE SIZE.

Refer to the section 'Release Restrictions' in this document before using this function.

COMMAND FORMAT:

```
EXSERV { SYSDATA } [ ( rtn-uid [rtn-slv [rtn-stg]] [] ]]
       { SD      }
```

rtn-uid - Variable name in which user ID is to be returned.
rtn-slv - Variable name in which current system level is to be returned. Value format is 'vv.mm.ll' where:
  vv - System version level.
  mm - System modification level.
  ll - Program change level.
rtn-stg - Variable name in which virtural machine storage size is to be returned. Value is in 'K'.

RETURN CODES:

0 -- SYSTEM DATA RETURNED.
4 -- CURRENT VM/370 SYSTEM DOES NOT SUPPORT THE 'EXTRACT SYSTEM DATA' (DIAG CODE 0) FUNCTION.

```
&CONTROL OFF
EXSERV SYSDATA (% SYSTEM)
&IF &RETCODE NE 0 &EXIT &RETCODE
&TYPE THE SYSTEM IS VM/370 RELEASE &SYSTEM
&EXIT &RETCODE
```

```
exec whatsys
THE SYSTEM IS VM/370 RELEASE 03.01.00
R;
```

Sample EXEC (WHATSYS) which identifies the system.

SYSGLBL (Global variable processor)

FUNCTION:

This routine is used to move global variable values to local variable values, set, or, test global variable values.

COMMAND FORMAT:

SET OPERATION:

```
EXSERV SYSGLBL gbl-var ( value [) ]
```

MOVE SYSTEM VARIABLE VALUE TO LOCAL VARIABLE VALUE:

```
EXSERV SYSGLBL gbl-var ...
```

TEST SYSTEM VARIABLE VALUE:

```
EXSERV SYSGLBL gbl-var ( opr value [) ]
```

RELEASE OPERATION:

```
EXSERV SYSGLBL ( RELEASE [) ]
```

gbl-var - Name of global variable.
opr - Compare operation (LT|LE|EQ|GE|GT|NE)
- Value to be used for compare or value used for set operation.

COMMENTS:

This function uses the area 'NUCRSV1' in the CMS nucleus.

RETURN CODES:

SET OPERATION:

ALWAYS ZERO.

MOVE GLOBAL VARIABLE VALUE TO LOCAL VARIABLE VALUE:

0 -- ALL LOCAL VARIABLES HAVE BEEN SET.
4 -- ONE OR MORE GLOBAL VARIABLES HAVE NOT BEEN DEFINED. LOCAL VARIABLE SET TO '%'.

TEST GLOBAL VARIABLE VALUE:

0 -- TEST CONDITION IS TRUE.
4 -- TEST CONDITION IS FALSE.
NOTE- A test for an undefined variable is always

false unless it is being tested for a null (%) value.

RELEASE OPERATION:

ALWAYS ZERO.

NOTES:

The System Global processor is very useful in maintaining a list of variables to be kept from one EXEC procedure to another. Thus, the variables can be, in effect, global to the user's terminal session.

To use a global variable, it must be defined by using the set operation. Thereafter, the variable exists as long as the user remains in CMS. Since the system global variable list is separate from the local variable list, there can be no name conflicts between locals and globals. By using the move operation, global and local variables of the same name will then have the same value. This does not mean, however, that changing this local variable will affect the global one. Globals can only be changed by the set operation. If the user wishes only to test the value of a system global variable, then the test operation is used and no local variable is created. When there is no further need for the system global variables, then the release operation releases the storage for all the variables. To release specific global variables, set the variable to a null value (%).

TSTOPT (Test option list)

FUNCTION:

This routine is used to test the options of the EXEC argument list for one of a mutually exclusive set of values. If none are found, a default value may be supplied. Additionally, short forms of the option values may be specified if desired. If an option value is found in the argument list, the value is returned in a local EXEC variable. If a short form is found, the long form of the option value is returned.

COMMAND FORMAT:

```
EXSERV { TSTOPT } rtn-var  [  ( value [minval] ... [) ]]
       { TO     }
```

rtn-var - Name of variable in which to return option value or default value.

value - Value for the EXEC option list search. Only one such value may occur in the option field of the EXEC argument list. The first value provided is considered the default value. If no default value is desired, a '%' may be specified.

minval - Number which indicates the minimum length needed to be considered an option value match. Default is the value length. eg., If 'PRINT 2' were specified, then 'PR', 'PRI', 'PRIN', and 'PRINT' would be considered as a match. 'PRT' would not be considered a match. Caution should be used in specifying too short a truncation that causes the set to be non-exclusive. For example, 'PRINT 1 PUNCH 1'.

RETURN CODES:

0 -- TEST SUCCESSFUL. OPTION VALUE OR DEFAULT VALUE RETURNED.

4 -- MORE THAN ONE OPTION VALUE FOUND IN LIST. DEFAULT VALUE RETURNED.

EXAMPLE:

Suppose an EXEC procedure called 'FOO' contained the following statement:

```
EXSERV TSTOPT WHERE (PRINTER 2 DISK PUNCH 3 TERMINAL 4)
```

and the variable '&WHERE' is used to control where the output is to be directed. Then when 'FOO' is invoked as follows, the option list can easily be checked and verified.

```
exec foo ... ( ... pun ... )
```

After the execution of the above EXSERV command, &WHERE would contain the value 'PUNCH'.

WRITE (Create or update a CMS file)

FUNCTION:

The write processor is used to create a new file, write over any record in an existing file, or add a new record to the end of a CMS file. The write update is used to overlay specific columns of an existing record without disturbing the data in other columns of the record.

COMMAND FORMAT:

```
EXSERV { WRITE  }  fn ft [fm [rcd [ff [lrecl]]]][ ([$COL][cc]
       { WRT    }                                    var ...[])]
       { WRTUPD }
       { WRU    }
```

fn - File name to be used for write operation.

ft - File type to be used for write operation.

fm - File mode to be used for write operation. Can be omitted if writing to existing file.

rcd - Record number to write in file. Defaults to zero (add new record to file). If writing an existing record, the entire record will be written unless 'WRTUPD' is specified.

ff - File format of file to write. Default value is 'F' (fixed), or whatever the existing file format is. Only 'F' or 'V' may be specified.

lrecl - Length of record to write. Defaults to 80 or whatever the existing record length is.

$COL - Used in conjunction with the '$COL' option of the READ operation. Indicates that the column numbers the data is to be written in is contained in a variable preceded by '&$' and followed by the data's variable name.

cc - Column in which to write data. A numeric value in the list is always considered a column number. Numeric data must be contained in a variable.

var - Variable name containing the data to be written.

RETURN CODES:

6xx -- 'xx' IS ALL POSSIBLE RETURN CODES FROM THE 'FSWRITE' MACRO.

EXAMPLE:

If the '$COL' option is used in conjunction with column numbers, then the precedence is as follows:
The processor first searches for an associated column variable (preceded by '&$'). If none is found, then the data is positioned at either the column specified or one column to the right of the last data item written. If this is the first data item and no column number was specified, the output is placed in column one.

Suppose the following statement was executed:

```
EXSERV WRITE MYFILE DATA A ($COL A B 25 C 37 D E )
```

And the variables had the following values:

```
&A = AAAAA       &$A = 10
&B = BBB         &$B - no value
&C = CCCCCCCC    &$C - no value
&D = DD          &$D = 20
&E - no value
```

The following record would be written:

```
col 10    16  20   25
    |     |   |    |
    AAAAA BBB DD E CCCCCCCC
```

Variable Length Format Files:
When adding new records to an existing variable length file, the maximum length record which may be written is equal to the length of the longest record in the file unless an explicit length is provided in the EXSERV command. If a record exceeds the maximum length, the record is truncated and no error is indicated.

When rewriting a variable length record, the existing record length cannot be changed. If the new record is longer then the existing record, the record will be truncated and no error will be indicated. If the new record is shorter, it is padded with blanks (WRITE operation) or with whatever data previously existed in the record (WRTUPD operation).

SH20-1922-0

Virtual Machine/370 Conversational Monitor System (VM/CMS) EXEC Language
Extensions Program Description/Operations Manual Printed in U.S.A. SH20-1922-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Virtual Machine/370 Conversational Monitor System (VM/CMS) EXEC Language Extensions
Program Description/Operations Manual

SH20-1922-0

READER'S COMMENT FORM

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

_____

_____

_____

_____

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

What is your occupation? _____

Number of latest Newsletter associated with this publication: _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Your comments, please . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold and tape | **Please Do Not Staple** | Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

International Business Machines Corporation
Department 68Y
P.O. Box 2750
225 John W. Carpenter Freeway, East
Irving, Texas 75062

Fold and tape | **Please Do Not Staple** | Fold and tape

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Cut or Fold Along Line

Virtual Machine/370 Conversational Monitor System (VM/CMS) EXEC Language Extensions Program Description/Operations Manual Printed in U.S.A. SH20-1922-0