IBM

GC24-5406-00

Virtual Machine/System Product Program Update Information

Restructured Extended Executor Language Enhancements

VM/SP Release 6
APARS VM36993, VM36994 and VM36988

IBM

GC24-5406-00

Virtual Machine/System Product Program Update Information

Restructured Extended Executor Language Enhancements

VM/SP Release 6
APARS VM36993, VM36994 and VM36988

First Edition (August 1989)

This edition, GC24-5406-00, applies to Release 6 of Virtual Machine/System Product (VM/SP), program number 5664-167, and to all subsequent releases of this product until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, 4300, and 9370 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

Ordering Publications

Publications are *not* stocked at the address given below. Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Dept. G60, P.O. Box 6, Endicott, NY, U.S.A. 13760. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

Who is this Book for?

This book is written for the systems programmer, application programmer, and others interested in an overview of the Restructured Extended Executor Language (REXX) enhancements. It is not a permanent addition to your library. Use it along with your current VM/SP manuals until otherwise indicated in new editions.

What Is This Book For?

This book is intended to help you use the Restructured Extended Executor (REXX) Language enhancements. It primarily contains general-use programming interfaces, which allow you to write programs that use the services of VM/SP. The programming interfaces are indicated as follows:

GPI

GPI end

Internal Product Information

Internal product information is provided as additional guidance for planning. This internal information should never be used as programming interface information.

IBM issues books such as this one to reflect software enhancements or software support for new hardware released on a program update tape (PUT). The books vary in size depending on the enhancement. The book typically describes the support, explains how to use it, and lists new messages, codes, and design changes. We recommend that you file it along with your VM/SP library in the binder provided for these books. If you do not have this binder, you can order it by using order number SX24-5226.

What Information does this Book Supplement?

This book supplements the information in the following VM/SP Release 6 manuals:

VM/SP Release 6 Books	Order Number
Application Development Reference for CMS	SC24-5284
Group Control System Command and Macro Reference	SC24-5250
System Messages and Codes	SC19-6204
System Product Interpreter Reference	SC24-5239

What Other Books Should I Have?

See the "Bibliography" on page 45 for a list of the program update books that have been issued since VM/SP Release 4 became available.

Contents

Overview . . . 1
  Programming Interfaces . . . 2

Modified Instructions . . . 3
CALL . . . 3
SIGNAL . . . 7

New Options . . . 9
Conditions and Condition Traps . . . 9

New Function . . . 14
CONDITION . . . 14

New Macros . . . 16
EPLIST . . . 16
GCSCALL . . . 20
RXITDEF . . . 22
RXITPARM . . . 23

Modified System Interface . . . 26
EXECCOMM . . . 26
  Function Codes (SHVCODE) . . . 26

Modified CSL Routine . . . 27
DMSCGS - Get Special REXX Values . . . 27

New System Exits . . . 29
System Exits . . . 29
  Invocation of the Language Processor by an Application Program . . . 29
  Invocation of the System Exits by the Language Processor . . . 31

New GCS ABEND Codes . . . 41
  GCS ABEND Codes . . . 41

Design Changes . . . 43
  Changed Modules (CMS) . . . 43
  Changed Modules (GCS) . . . 43
  New Modules (GCS) . . . 43
  Changed Modules (REXX) . . . 43
  New Modules (REXX) . . . 43
  Changed Macros (CMS) . . . 43
  New Macros (CMS) . . . 43
  Changed Macros (GCS) . . . 44
  Changed Macros (REXX) . . . 44
  New Macros (REXX) . . . 44
  Changed Control Blocks (CMS) . . . 44
  Changed Control Blocks (GCS) . . . 44
  Changed Execs (CMS) . . . 44
  Changed Execs (GCS) . . . 44
  Changed HELP Files . . . 44
  New HELP Files . . . 44

Bibliography 45

Overview

This book explains the enhancements to REXX for both the CMS and GCS environments. Additions to REXX are as follows:

- Condition trap option of the CALL instruction

  The CALL instruction also controls the trapping of certain conditions; that is, condition traps can modify the explicit flow of a program by specifying the ON or OFF keyword of the CALL instruction. This support is similar to the existing SIGNAL instruction.

- New keyword for CALL and SIGNAL instructions

  The syntax for specifying condition traps has changed. The new syntax includes the NAME keyword (NAME *trapname*). Both the CALL and SIGNAL instructions reflect this change.

- CONDITION built-in function

  A new built-in function, CONDITION, returns condition information associated with the current trapped condition.

- Argument limit increased for CALL instruction

  The number of argument strings on a CALL instruction has been increased to 20.

- Fetch Private Argument List function of EXECCOMM

  The Fetch private operation of the EXECCOMM interface is extended to return program supplied information about the following:

  Number of argument strings supplied to the program
  Nth argument string supplied to the program.

- User exits for system services

  User exits are now provided to let applications tailor the Procedures Language environment. The exit types are as follows:

Initialization/Termination	routines called at start up and termination of a program
System services	routines called to provide host environment services to the interpreter.

- PARSE VERSION update

  To reflect the changes made by this support, a value of 3.46 is returned by PARSE VERSION. The date returned is "31 May 1988".

- New GCS macros

  A new GCS macroinstruction, GCSCALL, is provided to allow invoking either the GCS REXX interpreter or the EXECCOMM interface directly from the user program.

  A new GCS macroinstruction, EPLIST, is provided to generate a DSECT for the GCS extended parameter list.

- New RXITDEF macro

  A new CMS/GCS macroinstruction, RXITDEF, is provided to assign the correct values to the symbols used for the exit routine function and subfunction codes.

- New RXITPARM macro

  A new CMS macroinstruction, RXITPARM, is provided to map the parameter list used to pass information between the language processor and an exit routine.

Programming Interfaces

The CALL instruction, SIGNAL instruction, CONDITION function, EPLIST macro, GCSCALL macro, RXITDEF macro, RXITPARM macro, and the System Exits are part of the general programming interfaces for VM/SP.

See "CALL" on page 3, "SIGNAL" on page 7, and "Conditions and Condition Traps" on page 9 in this book and Chapter 3 of the *VM/SP System Product Interpreter Reference*, SC24-5239, for documentation on the CALL and SIGNAL instructions.

See "CONDITION" on page 14, "EPLIST" on page 16, "GCSCALL" on page 20, "RXITDEF" on page 22, "RXITPARM" on page 23, and "System Exits" on page 29 in this book for documentation on the CONDITION function, the GCSCALL macro, the RXITDEF macro, the RXITPARM macro, and the System Exits.

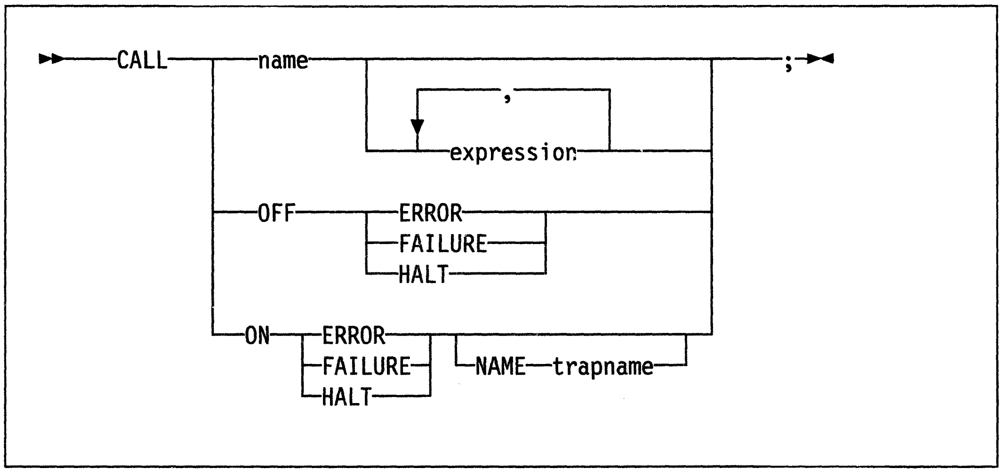
For more information on programming interfaces in VM/SP, see the *VM/SP Directory of Programming Interfaces for Customers* book, GC24-5417.

Modified Instructions

This section discusses modified instructions and should be used with the *VM/SP System Product Interpreter Reference*, SC24-5239. These modifications are marked by a vertical bar (|). Information concerning conditions and condition traps can be found in "Conditions and Condition Traps" on page 9.

CALL

GPI



Where:

name
: is a symbol or literal string that is taken as a constant.

OFF
: turns off the specified condition trap.

ON
: turns on the specified condition trap.

Note: For information on condition traps see "Conditions and Condition Traps" on page 9.

CALL is used to invoke a routine, or (if ON or OFF is specified) can be used to control the trapping of certain conditions.

When *name* is specified, CALL invokes a subroutine which can be:

- An internal routine
- An external routine
- A built-in function.

It can optionally return a result, and is functionally identical to the clause:

```
►►─result=name(─┬──────────────────────┬─)─;─►◄
                │ ┌─────────,─────────┐ │
                │ ▼                   │ │
                └───┴─expression─┴───────┘
```

except that the variable RESULT becomes uninitialized if no result is returned by the routine invoked.

The *name* given in the CALL instruction must be a valid symbol. If a string is used for *name* (that is, *name* is specified in quotes) the search for internal labels is bypassed, and only a built-in function or an external routine is invoked. Note that the names of built-in functions (and generally the names of external routines too) are in uppercase, and hence the name in the literal string should be in uppercase.

VM supports specifying up to 20 expressions, separated by commas. The expressions are evaluated in order from left to right, and form the argument string(s) during execution of the routine. Any ARG or PARSE ARG instructions, or ARG built-in function in the called routine will access these strings, rather than those previously active in the calling program. Expressions may be omitted, if appropriate, by including "extra" commas.

The CALL then causes a branch to the routine called *name* using exactly the same mechanism as function calls. The order in which these are searched for is described in the section on functions in the *VM/SP System Product Interpreter Reference*, SC24-5239, but briefly is as follows:

Internal routines:
These are sequences of instructions inside the same program, starting at the label that matches *name* in the CALL instruction. If the routine name is specified in quotes, then an internal routine will not be considered for that search order.

Built-in routines:
These are routines built in to the language processor for providing various functions. They always return a string containing the result of the function.

External routines:
Users can write or make use of routines that are external to the language processor and the calling program. An external routine can be written in any language, including REXX, which supports the system dependent interfaces. If an external routine, written in REXX, is invoked as a subroutine by the CALL instruction, any argument strings may be retrieved using the ARG or PARSE ARG instructions or the ARG built-in function.

During execution of an internal routine, all variables previously known are normally accessible. However, the PROCEDURE instruction may be used to set up a local variables environment to protect the subroutine and caller from each other. The EXPOSE option on the PROCEDURE instruction can be used to expose selected variables to a routine.

Calling an external program as a subroutine is similar to calling an internal routine. The external routine, however, is an implicit PROCEDURE in that all the caller's variables are always hidden and the status of internal values (NUMERIC settings, etc.) start with their defaults (rather than inheriting those of the caller).

When control reaches the internal routine, the line number of the CALL instruction is available in the variable SIGL (in the caller's variable environment). This may be used as a debug aid, as it is therefore possible to find out how control reached a routine. Note that if the internal routine uses the PROCEDURE instruction, then it will need to EXPOSE SIGL to get access to the line number of the CALL.

Eventually the subroutine should execute a RETURN instruction, and at that point control will return to the clause following the original CALL. If the RETURN instruction specified an expression, the variable RESULT will be set to the value of that expression. Otherwise, the variable RESULT is dropped (becomes uninitialized).

An internal routine can include calls to other internal routines, as well as recursive calls to itself.

Example:

```
/* Recursive subroutine execution... */
arg x
call factorial x
say x'! =' result
exit

factorial: procedure     /* calculate factorial by..  */
  arg n                  /*  .. recursive invocation. */
  if n=0 then return 1
  call factorial n-1
  return  result * n
```

During internal subroutine (and function) execution, all important pieces of information are automatically saved and are then restored upon return from the routine. These are:

- **The status of DO loops and other structures** — Executing a SIGNAL while within a subroutine is "safe" in that DO loops, etc., that were active when the subroutine was called are not deactivated (but those currently active within the subroutine will be deactivated).
- **Trace action** — Once a subroutine is debugged, you can insert a TRACE Off at the beginning of it, and this will not affect the tracing of the caller. Conversely, if you only wish to debug a subroutine, you can insert a TRACE Results at the start and tracing will automatically be restored to the conditions at entry (for example, "Off") upon return. Similarly, ? (interactive debug) and ! (command inhibition) are saved across routines.
- **NUMERIC settings** (the DIGITS, FUZZ, and FORM of arithmetic operations, described in the **NUMERIC** instruction in the *VM/SP System Product Interpreter Reference*, SC24-5239) are saved and are then restored on RETURN. A subroutine can therefore set the precision, etc., that it needs to use without affecting the caller.
- **ADDRESS settings** (the current and secondary destinations for commands — see the ADDRESS instruction in the *VM/SP System Product Interpreter Reference*, SC24-5239) are saved and are then restored on RETURN.

- **Condition traps** (CALL ON and SIGNAL ON) are saved and then restored on RETURN. This means that CALL ON, CALL OFF, SIGNAL ON, and SIGNAL OFF can be used in a subroutine without affecting the conditions set up by the caller.
- **Condition information** — This is the information returned by the CONDITION built-in function (see "CONDITION" on page 14).
- **Elapsed-time clocks** — A subroutine inherits the elapsed-time clock from its caller (see the TIME function in the *VM/SP System Product Interpreter Reference*, SC24-5239), but since the time clock is saved across routine calls, a subroutine or internal function can independently restart and use the clock without affecting its caller. For the same reason, a clock started within an internal routine is not available to the caller.
- **OPTIONS ETMODE/EXMODE** are saved and are then restored on RETURN. For more information — see the OPTIONS instruction in the *VM/SP System Product Interpreter Reference*, SC24-5239.

Implementation maximum: The total nesting of control structures, which includes internal routine calls, may not exceed a depth of 250.

GPI end

SIGNAL

GPI

```
►►─SIGNAL──┬─labelname──────────────────────────────────┬─;─►◄
           ├─┬───────┬─expression───────────────────────┤
           │ └─VALUE─┘                                   │
           ├─OFF────────┬─ERROR──────┬──────────────────┤
           │            ├─FAILURE────┤                   │
           │            ├─HALT───────┤                   │
           │            ├─NOVALUE────┤                   │
           │            └─SYNTAX─────┘                   │
           │                                             │
           └─ON───┬─ERROR────┬─┬──────────────────┬─────┘
                  ├─FAILURE──┤ └─NAME──trapname───┘
                  ├─HALT─────┤
                  ├─NOVALUE──┤
                  └─SYNTAX───┘
```

Where:

labelname
is a symbol or literal string that is taken as a constant.

OFF
turns off the specified condition trap.

ON
turns on the specified condition trap.

Note: For information on condition traps see "Conditions and Condition Traps" on page 9.

The SIGNAL instruction causes an **abnormal** change in the flow of control, or (if ON or OFF is specified) controls the trapping of certain conditions.

When neither ON nor OFF is specified, a label name is derived from *labelname* or taken from the result of evaluating the expression following VALUE. This must be a symbol, which is treated literally, or a literal string. The subkeyword VALUE may be omitted if *expression* does not begin with a symbol or literal string (i.e. if it starts with a special character, such as an operator or parenthesis). All active pending DO, IF, SELECT, and INTERPRET instructions in the current routine are then terminated (that is, they cannot be reactivated). Control then passes to the first label in the program that matches the required string, as though the search had started from the top of the program. If *labelname* is a symbol, the match is done independently of alphabetic case, but otherwise the label must match exactly.

Example:

```
Signal fred;  /* Jump to label "FRED" below */
  ....
  ....
Fred: say 'Hi!'
```

Because the search effectively starts at the top of the program, control will always pass to the first occurrence of the label in the program if duplicates are present.

When control reaches the specified label, the line number of the SIGNAL instruction is assigned to the special variable SIGL. This can be used to aid debugging, as it can be used to determine the source of a jump to a label.

Using SIGNAL with the INTERPRET Instruction

If, as the result of an INTERPRET instruction, a SIGNAL instruction is issued or a trapped event occurs, the remainder of the string(s) being interpreted will not be searched for the given label. In effect, labels within interpreted strings are ignored.

GPI end

New Options

Conditions and Condition Traps

GPI

CALL and SIGNAL modify the flow of execution in a REXX program by using condition traps. Condition traps are turned on or off using the ON or OFF subkeywords of the SIGNAL and CALL instructions (see “CALL” on page 3 and “SIGNAL” on page 7).

```
►►──┬─CALL───┬──┬─OFF──condition──────────────────────┬──;──►◄
    └─SIGNAL─┘  └─ON───condition──┬──────────────────┬┘
                                  └─NAME──trapname───┘
```

```
where condition and trapname are single symbols which are taken as constants.
```

Following one of these instructions, a condition trap is set to either ON (enabled) or OFF (disabled). The initial setting for all condition traps is OFF.

If a condition trap is enabled and the specified condition occurs, control passes to the routine or label *trapname*. SIGNAL or CALL is used, depending on whether the most recent trap for the condition was set using SIGNAL ON or CALL ON respectively.

The conditions and their corresponding events, which can be trapped are:

ERROR
is raised if any host command indicates an error condition upon return. It is also raised if any host command indicates failure and neither CALL ON FAILURE nor SIGNAL ON FAILURE are set. The condition is raised at the end of the clause that invoked the command, but will be ignored if the ERROR condition trap is already in the delayed state.

In VM (and TSO/E), when the RXCMD exit is not being used, CALL ON ERROR and SIGNAL ON ERROR trap all positive return codes; and will trap negative return codes if neither CALL ON FAILURE nor SIGNAL ON FAILURE are set. When the RXCMD exit is being used, CALL ON ERROR and SIGNAL ON ERROR will trap the RXCFERR flag returned by the RXCMD exit; and will trap the RXCFFAIL flag if neither CALL ON FAILURE nor SIGNAL ON FAILURE are set.

FAILURE
is raised if any host command indicates a failure condition upon return, but will be ignored if the FAILURE condition trap is already in the delayed state.

In VM (and TSO/E), when the RXCMD exit is not being used, CALL ON FAILURE and SIGNAL ON FAILURE trap all negative return codes from commands. When the RXCMD exit is being used, CALL ON FAILURE and SIGNAL ON FAILURE will trap the RXCFFAIL flag returned by the RXCMD exit.

HALT
is raised if an external attempt is made to interrupt execution of the program. The condition is raised at the end of the clause that was being interpreted when the interruption took place.

In VM, when the RXHLT exit is not being used, the CMS immediate command HI (Halt Interpretation) will cause a halt condition. If the RXHLT exit is being used, a halt condition will be recognized when the RXHFHALT flag is set by the exit routine. Refer to "Interrupting Execution and Controlling Tracing" in the *VM/SP System Product Interpreter Reference*, SC24-5239.

NOVALUE
raised if an uninitialized variable is used:

- As a term in an expression
- As the name following the VAR subkeyword of the PARSE instruction
- As an unassigned variable pattern in a parsing template.

This condition may only be specified for SIGNAL ON.

SYNTAX
raised if an interpretation error is detected. This condition may only be specified for SIGNAL ON.

Any ON or OFF reference to a condition trap replaces the previous state (ON or OFF, and any trap name) of that condition trap. Thus, a SIGNAL ON HALT replaces any current CALL ON HALT, and so on.

Action Taken When a Condition is Trapped

When a condition trap is currently enabled (ON has been specified), the trap is in effect. So, when the specified condition occurs, instead of the usual flow of control a "CALL *trapname*" or "SIGNAL *trapname*" is executed automatically (i.e., passes control to a label or routine). The label or routine given control will depend on whether you used the NAME *trapname* option when you enabled the condition trap.

If no explicit *trapname* was specified, control is passed to the label or routine that matches the name of the *condition* itself (ERROR, FAILURE, HALT, NOVALUE, or SYNTAX).

If *trapname* was specified following the NAME subkeyword of the CALL ON or SIGNAL ON instruction, control is passed to the label or routine specified, rather than the name of the *condition*.

The sequence of events, once a condition has been trapped, varies depending on whether a SIGNAL or CALL is executed:

- If the action taken is a SIGNAL, execution of the current instruction ceases immediately, the condition is disabled (set to OFF), and the SIGNAL takes place in exactly the same way as usual (see page 7).

  If any new occurrence of the condition is to be trapped, a new CALL ON or SIGNAL ON instruction for the condition is required to re-enable it once the label is reached. For example, if SIGNAL ON SYNTAX is enabled when a SYNTAX condition occurs, then if the SIGNAL ON SYNTAX label name is not found a normal syntax error termination will occur.

- If the action taken is a CALL, the CALL is made in the usual way (see page 3) except that the special variable RESULT is not affected by the call. If the routine should RETURN any data, then the returned character string is ignored.

Note that CALL ON can only occur at clause boundaries. Because these conditions (ERROR, FAILURE, and HALT) can arise during execution of an INTERPRET instruction, execution of the INTERPRET may be interrupted and later resumed if CALL ON was used.

Before the CALL is made, the condition trap is put into a *delayed* state. This state persists until the RETURN from the CALL, or until an explicit CALL (or SIGNAL) ON (or OFF) is made for the condition. This delayed state prevents a premature condition trap at the start of the routine called to process a condition trap. When a condition trap is in the delayed state it remains enabled, but if the condition is trapped again any action (including the updating of the condition information) will be delayed until one of the following events:

1. A CALL ON or SIGNAL ON, for the delayed condition, is executed. In this case a CALL or SIGNAL will take place immediately after the new CALL ON or SIGNAL ON instruction has been executed.
2. A CALL OFF or SIGNAL OFF, for the delayed condition, is executed. In this case the condition trap is disabled and the default action for the condition will occur at the end of the CALL OFF or SIGNAL OFF instruction.
3. A RETURN is made from the subroutine. In this case the condition trap is no longer delayed and the subroutine will be called again immediately.

On RETURN from the CALL, the original flow of execution is resumed (that is, the flow is not affected by the CALL).

Notes:

1. In all cases, the condition will be raised (and the current instruction terminated) immediately upon detection of the error. Therefore, the instruction during which an event occurs may be only partly executed. For example, if SYNTAX is raised during the evaluation of the expression in an assignment, the assignment will not take place. Note that ERROR, FAILURE, and HALT can only occur at clause boundaries, but could arise in the middle of an INTERPRET instruction.
2. The state (ON, OFF, or DELAY, and any *trapname*) of each condition trap is saved on entry to a subroutine and is then restored on RETURN. This means that CALL ON, CALL OFF, SIGNAL ON, and SIGNAL OFF can be used in a subroutine without affecting the conditions set up by the caller. See the CALL instruction (page 3) for details of other information that is saved during a subroutine call.
3. The state of condition traps is not affected when an external routine is invoked by a CALL, even if the external routine is a REXX program. On entry to any REXX program, all condition traps have an initial setting of OFF.
4. While user input is executed during interactive tracing, all conditions are set OFF so that unexpected transfer of control does not occur should (for example) the user accidentally use an uninitialized variable while SIGNAL ON NOVALUE is active. For the same reason, a syntax error during interactive tracing will not cause exit from the program, but is trapped specially and then ignored after a message is given.
5. Certain execution errors are detected by the host interface either before execution of the program starts or after the program has exited. These errors cannot be trapped by SIGNAL ON SYNTAX.

Note that **labels** are clauses consisting of a single symbol followed by a colon. Any number of successive clauses can be labels; therefore, multiple labels are allowed before another type of clause.

Condition Information

When any condition is trapped and causes a SIGNAL (or CALL), this becomes the current trapped condition, and certain condition information associated with it is recorded. This information can be inspected by using the CONDITION built-in function

The condition information includes:

- The name of the current trapped condition
- Any descriptive string associated with that condition
- The instruction executed as a result of the condition trap (CALL or SIGNAL)
- The status of the trapped condition.

The descriptive string varies, depending on the condition trapped. In the case of SIGNAL, the descriptive string that is passed to the external environment as command results in one of the following:

ERROR	The string that was processed and resulted in the error condition.
FAILURE	The string that was processed and resulted in the failure condition.
HALT	Any string associated with the halt request. This can be the null string if no string was provided.
NOVALUE	The derived name of the variable whose attempted reference caused the NOVALUE condition.
SYNTAX	Any string associated with the error by the language processor. This can be the null string if no specific string is provided. Note that the special variable RC and SIGL provide information on the nature and position of the processing error.

The current condition information is replaced when control is passed to a label as the result of a condition trap (CALL ON or SIGNAL ON). Condition information is saved and restored across subroutine or function calls, including one due to a CALL ON trap. A routine invoked by a CALL ON, therefore, can access the appropriate condition information. Any previous condition information is still available after the routine returns.

The Special Variable SIGL

When any transfer of control due to a SIGNAL (or CALL) takes place, the line number of the clause currently executing is stored in the REXX special variable SIGL. This is especially useful for SIGNAL ON SYNTAX when the number of the line in error can be used, for example, to control an editor. Typically, code following the SYNTAX label may PARSE SOURCE to find the source of the data, then invoke an editor to edit the source file positioned at the line in error. Note that in this case the program has to be reinvoked before any changes made in the editor can take effect.

Alternatively, SIGL can be used to help determine the cause of an error (such as the occasional failure of a function call) as in the following example:

```
/* Standard handler for SIGNAL ON SYNTAX */
syntax:
  errormsg='REXX error' rc 'in line' sigl':' errortext(rc)
  say errormsg
  say sourceline(sigl)
  trace '?r'; nop
```

This code first displays the error code, line number, and message text. It then displays the line in error, and finally drops into debug mode to let you to inspect the values of the variables used at the line in error.

The Special Variable RC

For ERROR and FAILURE, the REXX special variable RC is set to the command return code error number before control is transferred to the condition label. For SIGNAL ON SYNTAX, RC is set to the syntax error number.

GPI end

New Function

CONDITION

GPI

```
►►──CONDITION(option)──────►◄
```

returns the condition information associated with the current trapped condition. See "CALL" on page 3 and "SIGNAL" on page 7 for a description of condition traps. Four pieces of information can be requested:

- The name of the current trapped condition
- Any descriptive string associated with that condition
- The instruction executed as a result of the condition trap (SIGNAL or CALL)
- The status of the trapped condition.

The following *option* can be supplied to select the requested information. Only the first letter is significant.

`Condition name`	returns the name of the current trapped condition.
`Description`	returns any descriptive string associated with the current trapped condition. See page 12 for the list of possible strings. If no description is available, a null string is returned.
`Instruction`	returns the keyword for the instruction executed when the current condition was trapped, being either CALL or SIGNAL. This is the default if *option* is not specified.
`Status`	returns the status of the current trapped condition. This can change during execution, and is either: ON - the condition is enabled; OFF - the condition is disabled; DELAY - any new occurrence of the condition is delayed.

If no condition has been trapped (that is, there is no current trapped condition) then the CONDITION function returns a null string in all four cases.

Here are some examples:

```
CONDITION()          ->    'CALL'        /* perhaps */
CONDITION('C')       ->    'FAILURE'
CONDITION('I')       ->    'CALL'
CONDITION('D')       ->    'FailureTest'
CONDITION('S')       ->    'OFF'        /* perhaps */
```

Note:

The condition information returned by the CONDITION function is saved and restored across subroutine calls (including those caused by a CALL ON condition trap). Therefore, once a subroutine invoked due to a CALL ON trap has returned, the current trapped condition reverts to the current condition before the CALL took place. CONDITION returns the values it returned before the condition was trapped.

GPI end

New Macros

This section discusses modified macros and should be used with the *VM/SP GCS Command and Macro Reference*, SC24-5250, and the *VM/SP Application Development Reference*, SC24-5284. These modifications are marked by a vertical bar (|).

EPLIST

GPI

Use the EPLIST macro to generate a DSECT for the GCS extended parameter list.

Format

[*label*]	**EPLIST**	

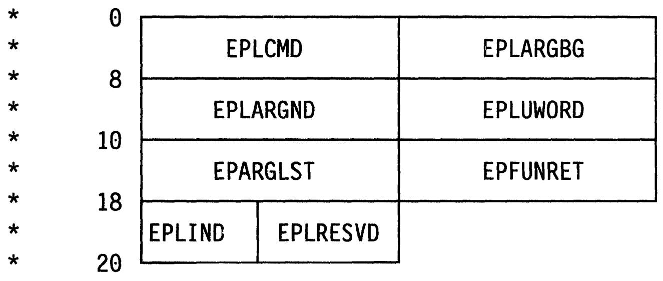
Optional Parameter

label
is an optional assembler label for the statement. The first statement in the EPLIST macro expansion is labeled EPLIST.

Usage Note

1. The EPLIST macroinstruction expands as follows:

```
         EPLIST
*
***      EPLIST - EXTENDED PLIST DSECT
*
```



```
*
***      EPLIST - EXTENDED PLIST DSECT
*
EPLIST   DSECT
EPLCMD   DS    A             ADDRESS OF COMMAND TOKEN.
EPLARGBG DS    A             ADDR OF BEGINNING OF ARGUMENTS.
EPLARGND DS    A             ADDR OF END OF ARGUMENTS.
EPLFBL   DS    A             ADDR OF THE FILE BLOCK
EPARGLST DS    A             ADDR OF THE FUNCTION ARGUMENT LIST
EPFUNRET DS    A             ADDR FOR RETURN OF FUNCTION DATA
EPL4LNBY EQU   *-EPLIST       4 WORD HEADER LENGTH IN BYTES
EPL4LNDW EQU   (*-EPLIST+7)/8 4 WORD HEADER LENGTH IN DWORDS
***************************************************************
* While the preceding 6 words are the same for both CMS and
* GCS, the following word is applicable for GCS only.
***************************************************************
```

Figure 1 (Part 1 of 3). EPLIST Control Block Format

```
EPLIND   DS    X             Indicator byte.
EPLPGM   EQU   X'00'         Program issued command.
EPLACMD  EQU   X'01'         Called from System Product
*                              Interpreter with ADDRESS COMMAND
EPLFNC   EQU   X'05'         Subroutine or function call.
EPLCONS  EQU   X'0B'         Console command.
         DS    3X            Reserved
         SPACE ,
EPLLENBY EQU   *-EPLIST       Total length in bytes.
EPLLENDW EQU   (*-EPLIST+7)/8 Total length in dwords.
         EJECT ,
****************************************************************
* The following equates for byte 0 of reg 1 are applicable
* for CMS only.  GCS uses the EPLIND byte instead.  This is
* defined above.  The following equates are retained to allow
* programs to be compiled on CMS and execute in either CMS or
* GCS environments.
****************************************************************
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                               *
*        THE EXTENDED PLIST FLAGS INDICATE THE PRESENCE         *
*        OF AN EXTENDED PLIST IN REGISTER 0. THE HIGH           *
*        ORDER BYTE OF REGISTER 1 WILL CONTAIN EITHER           *
*        EPLCMDFL OR EPLFNCFL TO INDICATE THE EXTENDED          *
*        PLIST IS AVAILABLE.  ONLY THE FIRST 4 WORDS OF         *
*        OF THE EXTENDED PLIST ARE AVAILABLE WITH THESE         *
*        CODES.                                                 *
*                                                               *
*        IF THE HIGH ORDER BYTE OF REGISTER 1 CONTAINS          *
*        EPFUNSUB, THEN THE INVOCATION IS AN EXTERNAL           *
*        FUNCTION/SUBROUTINE CALL FROM REXX.  WITH THIS         *
*        PLIST, ALL 6 WORDS OF THE PLIST ARE AVAILABLE.         *
*        WORD 5 POINTS TO A LIST OF DOUBLE WORD ADLENS          *
*        (ADDRESS-LENGTH PAIRS) WHICH DESCRIBE THE              *
*        ARGUMENTS TO THE ROUTINE (EPARGLST).  WORD 6           *
*        (EPFUNRET) IS THE LOCATION FOR THE CALLED              *
*        ROUTINE TO STORE THE ADDRESS OF AN EVALBLOK            *
*        TO RETURN DATA TO THE CALLING PROGRAM.                 *
*                                                               *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Figure 1 (Part 2 of 3). EPLIST Control Block Format

```
EPLCMDFL EQU   X'0B'        EXTENDED PLIST AVAILABLE FLAG.
EPLFNCFL EQU   X'01'        EXTENDED PLIST AVAILABLE FLAG.
EPFUNSUB EQU   X'05'        EXTERNAL FUNCTION PLIST AVAILABLE
*
* FLAG DEFINITIONS.  EXCEPT AS NOTED, ONLY THE FIRST FOUR
* WORDS OF THE EXTENDED PLIST ARE AVAILABLE.
*                         EPLIST
* FLAG         VALUE      AVAIL? MEANING
EPLFPROG EQU   X'00'           N PROGRAM
EPLFCMND EQU   X'01'           Y ADDRESS COMMAND
EPLFSBCM EQU   X'02'           Y SUBCOM
EPLFNNUE EQU   X'03'           Y NO NUCEXT, EXTENDED
EPLFNNUT EQU   X'04'           N NO NUCEXT, TOKENIZED
EPLFRXFN EQU   X'05'           Y REXX EXTERNAL FUNCTION,
*                                6 WORD EXTENDED PLIST PRESENT
EPLFIMMD EQU   X'06'           Y IMMEDIATE COMMAND
EPLFSRCH EQU   X'0B'           Y COMMAND SEARCH
EPLFENDC EQU   X'FE'           N END OF COMMAND
EPLFABEN EQU   X'FF'           N ABEND OR NUCXDROP
```

Figure 1 (Part 3 of 3). EPLIST Control Block Format

GPI end

GCSCALL

GPI

Use the GCSCALL macro to:

- call the GCS REXX interpreter directly from the user program
- call the EXECCOMM interface directly from the user program
- determine the address of the environment work block (WORKBLOK) for the current EXECCOMM variable pool
- change the address of the environment work block (WORKBLOK) for the current EXECCOMM variable pool.

Format

[*label*]	**GCSCALL**	$\left\{\begin{array}{l}\textbf{REXX}\\\textbf{EXECCOMM}\\\textbf{GETCOMM}\\\textbf{SETCOMM}\end{array}\right\}$

Optional Parameters

label
is an optional assembler label for the statement.

REXX
specifies a link to the GCS REXX Interpreter. You must set up the following registers in order to use this option:

R0	Pointer to the Extended Plist.
R1	Pointer to the Tokenized Plist.
R13	Pointer to a standard 18 fullword register save area.
R14	Reserved for module linkage.
R15	Reserved for module linkage and return code.

On return, registers 0 through 13 should be unchanged, register 14 will be the address to which control was returned, and register 15 will be the return code.

EXECCOMM
specifies a link to the GCS EXECCOMM interface. You must set up the following registers in order to use this option:

R0	Pointer to the shared variable block request chain.
R1	Ignored.
R13	Pointer to a standard 18 fullword register save area.
R14	Reserved for module linkage.
R15	Reserved for module linkage and return code.

On return, registers 0 through 13 should be unchanged, register 14 will be the address to which control was returned, and register 15 will be the return code.

GETCOMM
determines the address of the environment work block (WORKBLOK) for the current EXECCOMM variable pool. You must set up the following registers in order to use this option:

R0	Reserved for the returned WORKBLOK address.
R1	Ignored.
R13	Pointer to a standard 18 fullword register save area.
R14	Reserved for module linkage.
R15	Reserved for module linkage and return code.

On return, registers 1 through 13 should be unchanged, register 0 will contain the address of the current WORKBLOK, register 14 will be the address to which control was returned, and register 15 will be the return code. A return code of zero indicates the function was successful.

SETCOMM
changes the address of the environment work block (WORKBLOK) for the current EXECCOMM variable pool. You must set up the following registers in order to use this option:

R0	Address of the desired WORKBLOK.
R1	Ignored.
R13	Pointer to a standard 18 fullword register save area.
R14	Reserved for module linkage.
R15	Reserved for module linkage and return code.

On return, registers 0 through 13 should be unchanged, register 14 will be the address to which control was returned, and register 15 will be the return code. A return code of zero indicates the function was successful.

Usage Notes

1. For GCSCALL EXECCOMM, to convert an existing SVC call to a BALR call, replace the:

   ```
   <label>  EXECCOMM REQLIST=addr
   ```

   statement in your program with:

   ```
   <label>  LA    R0,addr
            GCSCALL EXECCOMM
   ```

   or equivalent statements.

2. In the case of GCSCALL REXX and GCSCALL EXECCOMM, when used from supervisor state, the calling program may be running either enabled for all interrupts, enabled for some and disabled for other interrupts, or disabled for all interrupts. The GCS REXX interpreter enables for all interrupts periodically during processing and returns to the calling program enabled for all interrupts.

3. The GCSCALL GETCOMM and GCSCALL SETCOMM functions are intended for use only within an exit routine. If the exit routine chooses to use SETCOMM to change the WORKBLOK pointer during the exit processing, the exit routine must first use GETCOMM to remember the original pointer value, and use SETCOMM to restore the pointer to this value before returning to the GCS REXX interpreter. The active WORKBLOK address must be consistent with the register contents returned to the interpreter.

GPI end

RXITDEF

GPI

Use the RXITDEF macro to assign the correct values to the symbols used for the exit routine function and subfunction codes. This macro may be used for CMS and GCS programs.

Format

	RXITDEF	

Usage Note

The following symbols are assigned by this macro:

```
RXFNC    EQU  X'0002'          Process a function request.
RXFNCCAL EQU          X'0001'  FNC Call a function/subroutine.
RXCMD    EQU  X'0003'          Process a command request.
RXCMDHST EQU          X'0001'  CMD Process a host command request.
RXMSQ    EQU  X'0004'          Manipulate the session queue.
RXMSQPLL EQU          X'0001'  MSQ Pull an entry from queue.
RXMSQPSH EQU          X'0002'  MSQ Push an entry onto queue.
RXMSQSIZ EQU          X'0003'  MSQ Determine the queue size.
RXSIO    EQU  X'0005'          Perform Session Input/Output.
RXSIOSAY EQU          X'0001'  SIO Output a SAY string.
RXSIOTRC EQU          X'0002'  SIO Output a TRACE string.
RXSIOTRD EQU          X'0003'  SIO Terminal Read.
RXSIODTR EQU          X'0004'  SIO Debug Terminal Read.
RXSIOTLL EQU          X'0005'  SIO Determine line length.
RXMEM    EQU  X'0006'          Memory management services.
RXMEMGET EQU          X'0001'  MEM Get memory.
RXMEMRET EQU          X'0002'  MEM Return memory.
RXHLT    EQU  X'0007'          Halt services.
RXHLTCLR EQU          X'0001'  HLT Clear the halt status.
RXHLTTST EQU          X'0002'  HLT Test the halt status.
RXTRC    EQU  X'0008'          Test the TRACE status.
RXTRCTST EQU          X'0001'  TRC Test the TRACE status.
RXINI    EQU  X'0009'          Initialization service.
RXINIEXT EQU          X'0001'  INI Initialization exit.
RXTER    EQU  X'000A'          Termination service.
RXTEREXT EQU          X'0001'  TER Termination exit.
```

GPI end

RXITPARM

GPI

Use the RXITPARM macro to map the parameter list used to pass information between the language processor and an exit routine. This macro may be used for CMS and GCS programs.

Format

	RXITPARM	

Usage Notes

The following symbols are defined by this macro:

```
RXITPARM DSECT ,
****************************************************************
*  The following parameters are common to all exit routines.
****************************************************************
RXIEXIT  DS    H       Exit code                         (input)
RXISUBFN DS    H       Exit subfunction                  (input)
RXIUSER  DS    F       User word                         (input)
RXICFLAG DS    X       Exit processing control flags
RXIFFLAG DS    X       Exit specific flags
RXIFEVAL EQU   X'01'   String returned via EVALBLOK     (output)
RXIPLEN  DS    H       Length of plist in bytes          (input)
         DS    F       Reserved for future use
RXITMAPX DS    CL24    Beginning of exit specific parameters
RXITMAPZ EQU   *       End of exit parameter list
RXITMAPL EQU   RXITMAPZ-RXIEXIT Length of the parameter list

****************************************************************
*  The following parameters are unique to the RXFNC exit.
****************************************************************
         ORG   RXITMAPX
RXFFERR  EQU   X'80'   Invalid call to routine          (output)
RXFFNFND EQU   X'40'   Routine not found                (output)
RXFFSUB  EQU   X'20'   Subroutine call                   (input)
RXFFNC   DS    A       Pointer to the routine name       (input)
RXFFNCL  DS    F       Length of the routine name        (input)
RXFARGS  DS    A       Pointer to argument list          (input)
RXFRET   DS    A       Pointer to EVALBLOK for
*                      function RETURN result           (output)
RXFPLEN  EQU   *-RXITMAPX

****************************************************************
*  The following parameters are unique to the RXCMD exit.
****************************************************************
         ORG   RXITMAPX
RXCFFAIL EQU   X'80'   Command FAILURE occurred         (output)
RXCFERR  EQU   X'40'   Command ERROR occurred           (output)
RXCADDR  DS    CL8     Current ADDRESS setting           (input)
RXCCMD   DS    A       Pointer to the command            (input)
RXCCMDL  DS    F       Length of the command             (input)
RXCRETC  DS    A       Pointer to return code buffer   (in+out)
RXCRETCL DS    F       Length of return code           (in+out)
RXCPLEN  EQU   *-RXITMAPX
```

```
****************************************************************
*  The following parameters are unique to the RXMSQ exit.
****************************************************************
         SPACE 1
*  The following parameters are used for the RXMSQPLL function.
         ORG   RXITMAPX
RXMFEMPT EQU   X'40'    Queue was empty                (output)
RXMRETC  DS    A        Pointer to return value buffer (in+out)
RXMRETCL DS    F        Length of return value         (in+out)
RXMPLLPL EQU   *-RXITMAPX
         SPACE 1
*  The following parameters are used for the RXMSQPSH function.
         ORG   RXITMAPX
RXMFLIFO EQU   X'80'    Stack the line LIFO             (input)
RXMVAL   DS    A        Pointer to line to stack        (input)
RXMVALL  DS    F        Length of line to stack         (input)
RXMPSHPL EQU   *-RXITMAPX
         SPACE 1
*  The following parameters are used for the RXMSQSIZ function.
         ORG   RXITMAPX
RXMQSIZE DS    F        Number of lines in stack       (output)
RXMSIZPL EQU   *-RXITMAPX

****************************************************************
*  The following parameters are unique to the RXSIO exit.
****************************************************************
         SPACE 1
*  The following parameters are used for the RXSIOTLL function.
         ORG   RXITMAPX
RXSSIZE  DS    F        Size of terminal in bytes      (output)
RXSSIZPL EQU   *-RXITMAPX
         SPACE 1
*  The following parameters are used for RXSIOSAY and RXSIOTRC.
         ORG   RXITMAPX
RXSVAL   DS    A        Address of line to display      (input)
RXSVALL  DS    F        Length of line to display       (input)
RXSOUTPL EQU   *-RXITMAPX
         SPACE 1
*  The following parameters are used for RXSIOTRD and RXSIODTR.
         ORG   RXITMAPX
RXSRETC  DS    A        Pointer to return value buffer (in+out)
RXSRETCL DS    F        Length of return value         (in+out)
RXSINPPL EQU   *-RXITMAPX

****************************************************************
*  The following parameters are unique to the RXMEM exit.
****************************************************************
         SPACE 1
*  The following parameters are used for RXMEMGET and RXMEMREL.
         ORG   RXITMAPX
RXMFLO24 EQU   X'80'    Storage must be allocated below
*                       the 16Mb line.                  (input)
RXMSSIZE DS    F        Size of storage (in double words)
*                       to be allocated or released     (input)
RXMADDR  DS    A        Address of storage allocated   (in-out)
*                       or being released
RXMPLEN  EQU   *-RXITMAPX
```

```
****************************************************************
*  The following parameters are unique to the RXHLT exit.
****************************************************************
         SPACE 1
*  The following parameters are used for RXHLTTST.
*  (No unique parameters are required for RXHLTCLR.
         ORG   RXITMAPX
RXHFHALT EQU   X'80'   HALT condition occurred         (output)
RXHSTR   DS    A       Pointer to EVALBLOK containing an
*                      optional HALT string            (output)
RXHPLEN  EQU   *-RXITMAPX

****************************************************************
*  The following parameters are unique to the RXTRC exit.
****************************************************************
         ORG   RXITMAPX
RXTFTRAC EQU   X'80'   External TRACE setting           (output)
RXTPLEN  EQU   *-RXITMAPX

****************************************************************
*  No unique parameters are used for the RXINI and RXTER exits.
****************************************************************
```

GPI end

Modified System Interface

This section discusses modified system interface and should be used with the *VM/SP System Product Interpreter Reference*, SC24-5239. These modifications are marked by a vertical bar (|).

EXECCOMM

GPI

Function Codes (SHVCODE)

P — Fetch private information. This interface is identical to the F fetch interface, except that the name refers to certain fixed information items that are available. Only the first letter of each name is checked (though callers should supply the whole name), and the following names are recognized:

PARM	Fetch the number of argument strings. The number of argument strings supplied to the program is placed in the caller's buffer. The number is formatted as a character string. **Note:** When specifying PARM, each letter **must** be supplied.
PARM.n	Fetch the Nth argument string. Argument string n is placed in the caller's buffer. If argument string n can not be supplied (whether omitted, null, or fewer than n argument strings specified), a null string is returned. PARM.1 returns the same result as ARG. **Note:** When specifying PARM.n, 'PARM.' **must** be supplied.

GPI end

Modified CSL Routine

This section discusses a modified CSL routine and should be used with the *VM/SP Application Development Reference for CMS*, SC24-5284. Specifically, the *varname* parameter definition to the DMSCGS routine has been modified while the syntax diagram has not changed. These modifications are marked by a vertical bar (|).

DMSCGS - Get Special REXX Values

GPI

Use the CSL routine DMSCGS to retrieve REXX EXEC arguments, source program, or version information.

Format

Call to DMSCSL	**DMSCGS** *,retcode,*	*varname* *,varname_length* *,varvalue* *,varvalue_buffer_length* *,varvalue_actual_length*

Parameters

Call to DMSCSL
is the language-dependent format for invoking a CSL routine. Refer to *VM/SP Application Development Reference for CMS*, SC24-5284, for examples of call formats.

DMSCGS
is the name of the CSL routine being invoked. The value DMSCGS can be passed directly or in a variable. Note that you must pad two blanks on the right because the CSL routine name must be eight characters in length.

retcode
is a signed integer variable, with a length of 4, to hold the return code from DMSCGS.

| *varname*
| specifies the special REXX information you want to obtain. The value ARG, PARM, PARM.*n*, SOURCE, or VERSION can be passed directly or in a variable. This field is used for input only. It must be a character variable.

| **Note:** Except for the PARM variable ('PARM' or 'PARM.*n*'), REXX uses only the first letter of this parameter name for comparisons. You should completely spell out the names of the REXX variables you are passing to avoid possible confusion.

varname_length
contains the length of the REXX variable's name. This field must be a signed integer variable, with a length of 4, and it is used for input only.

varvalue
is the name of a buffer used to return the REXX variable's value. This field is used for output only. It must be a character variable.

varvalue_buffer_length
contains the length of the buffer *varvalue*. This field must be a signed integer variable, with a length of 4, and it is used for input only.

varvalue_actual_length
contains the length of the data returned in the *varvalue* parameter. This field must be a signed integer variable, with a length of 4, and it is used as output only. (See Usage Note 2.)

Usage Notes

1. This routine provides the equivalent of the REXX statements PARSE ARG, PARSE SOURCE, and PARSE VERSION. See the *VM/SP System Product Interpreter Reference* for a description of the PARSE instruction.
2. If the length of the returned data (*varvalue_actual_length*) is shorter than the buffer space reserved (*varvalue_buffer_length*), the unused part of the buffer is unchanged.

   If the length of the returned data (*varvalue_actual_length*) is longer than the buffer space reserved (*varvalue_buffer_length*), the buffer is filled and a return code of 200 is set.

Return Codes

Note that for the last two return codes listed, the *nn* designates the relative position of the parameter: *retcode* is always parameter number 01, the next parameter is number 02, and so on.

Code	Meaning
0	Normal completion.
112	The number of parameters passed on the call was incorrect.
200	The data returned in *varvalue* has been truncated. (The *varvalue_actual_length* variable contains the length of the data before it was truncated.)
201	REXX is working and cannot share variables now.
202	REXX is not active.
207	Variable name is not one of the three supported. (This return code is also issued if DMSCGS is called from within an EXEC2 environment.)
208	Insufficient storage.
209	Storage failure (error in CMSSTOR or SUBPOOL macro).
10*nn*	The data type for parameter *nn* is incorrect.
20*nn*	The length for parameter *nn* is incorrect.

GPI end

New System Exits

This section discusses system exits and should be used with the *VM/SP System Product Interpreter Reference*, SC24-5239.

System Exits

This support provides a set of exits to the System Product Interpreter to allow applications to tailor the REXX environment. The exits fall into two categories:

Initialization/Termination	routines called at startup and termination of a program
System services	routines called to provide host environment services to the language processor.

These services are provided in both CMS and GCS. For the most part, the interfaces are identical between the two. The following description focuses on CMS, with the GCS differences noted.

Invocation of the Language Processor by an Application Program

The system exits are defined at language processor invocation by means of a specified FILEBLOK extension. The FILEBLOK extension contains a pointer in the 7th fullword of the extension block that points to the exit. The 8th fullword of the extension block is used to pass a user word value that is returned to the parameter list when an exit is entered.

Note: The FBLENAME portion of the FILEBLOK extension is not supported by GCS. The GCS FBLOCK macro reserves the 5th and 6th fullwords.

GPI

```
         MACRO
         FBLOCK
*
***      FBLOCK - EXEC FILE EXECUTION CONTROL BLOCK
*
*            0 +---------------------------------------+
*              |               FBLNAME                 |
*            8 +---------------------------------------+
*              |               FBLTYPE                 |
*           10 +---------+---------+-------------------+
*              | FBLMODE | FBLEXTL |     FBLDLS        |
*           18 +---------+---------+-------------------+
*              |      FBLDLE       |     FBLPREF       |
*           20 +-------------------+-------------------+
*              |    FBLPREF(cont'd)|     FBLENAME      |
*           28 +-------------------+-------------------+
*              |  FBLENAME(cont'd)|     FBLSEXIT      |
*           30 +-------------------+-------------------+
*              |     FBLEUSER      |
*           38 +-------------------+
*
*
***      FBLOCK - EXEC FILE EXECUTION CONTROL BLOCK
*
         SPACE 1
FBLOCK   DSECT
         DS    0F
FBLNAME  DS    CL8           Filename
FBLTYPE  DS    CL8           Filetype
FBLMODE  DS    CL2           Filemode
FBLLFI   EQU   *-FBLNAME     Length of fileid
FBLEXTL  DS    AL2(0)        Extension block length (words)
FBLDLS   DS    AL4           Descriptor list start
FBLDLE   DS    AL4           Descriptor list end
FBLPREF  DS    CL8           Explicit initial prefix
FBLENAME DS    CL8           Explicit environment name
FBLSEXIT DS    F             Pointer to exit vector
FBLEUSER DS    F             Userword for system exits
FBLLENL  EQU   *-FBLNAME     Length of FBLOCK (bytes)
FBLLEND  EQU   (FBLLENL+7)/8 Length of FBLOCK (dwords)
         EJECT
         POP   PRINT
         MEND
```

GPI end

The exit vector is a list of doubleword tokens, with a doubleword fence signaling the end of the list. Each token consists of a code in the first halfword identifying an exit and the address in the second fullword indicating the address of the exit. The second halfword of the doubleword token is reserved.

The following exits may be specified in the list. The RXITDEF macro establishes the equated values for each of these exit names, and for their associated subfunctions.

RXFNC Process external functions

RXCMD	Process host commands
RXMSQ	Manipulate session queue
RXSIO	Session I/O
RXMEM	Memory services
RXHLT	Halt processing
RXTRC	Test external trace indicator
RXINI	Initialization processing
RXTER	Termination processing.

Invocation of the System Exits by the Language Processor

Call Conditions

Note: If the language processor was invoked by a GCS problem state application, then the exits are entered in a problem state, enabled for interrupts, and enabled with the storage key of the original application program. If the language processor was invoked by a GCS supervisor state application, then the exits are entered in a supervisor state, key zero, enabled for interrupts. If the language processor was invoked by a CMS application, then the exits will be invoked in supervisor state, nucleus key, and enabled for interrupts.

The following registers are defined on entry:

Register 1	A pointer to the system exit parameter list. This parameter list varies with each entered exit. See below for details on the format of this parameter list for each exit.
Register 13	A pointer to a 20 fullword register save area.
Register 14	The return address.
Register 15	The entry point address.

The system exit parameter list consists of several fields that are commonly used by all the exits, followed by fields that are specific to each exit. The common information includes:

- RXIEXIT - the exit being invoked
- RXISUBFN - the subfunction requested for that exit
- RXIUSER - the optional fullword of user data
- RXICFLAG - a flag byte used to control exit processing
- RXIFFLAG - a flag byte used for exit specific communication
- RXIPLEN - the length of the parameter list.

See “System Exit Definitions” on page 32 for the format of the control blocks.

Return Conditions

On return from the exit, register 15 contains the exit return code and the parameter list is updated with the appropriate results. The return code in register 15 signals one of 3 actions:

RC=0	Successful handling of the service. The parameter list has been updated as appropriate for that exit.
RC=1	Exit chooses not to handle the service request. The language processor handles the request by the default means.

RC=-1 An irrecoverable error occurred during processing of this request. REXX error 48 (`Failure in system service`) is raised.

The exit routines must save registers 0-14 upon invocation, and restore them before returning to their caller.

System Exit Definitions

GPI

The RXITDEF macro is used to establish the equated values for each of the exit names, and for their associated subfunction names. The RXITPARM macro is used to establish the mapping DSECT for these parameter lists. The EPLIST and EVALBLOK mapping is further described in the *VM/SP System Product Interpreter Reference*, SC24-5239. Also, you may use the EPLIST and the EVALBLOK macros to provide the mapping for each of these DSECTS.

RXFNC Process external functions.

RXFNCCAL Call an external function.

```
RXIEXIT   DS   H       Exit code = 2
RXISUBFN  DS   H       Exit subfunction = 1
RXIUSER   DS   F       User word
RXICFLAG  DS   X       Exit processing control flags
RXIFFLAG  DS   X       Exit specific flags
RXFFERR   EQU  X'80'   Incorrect call to routine
RXFFNFND  EQU  X'40'   Routine not found
RXFFSUB   EQU  X'20'   Subroutine call
RXIPLEN   DS   H       Length of plist in bytes
RXIRESRV  DS   F       Reserved for future use
RXFFNC    DS   A       Pointer to the routine name
RXFFNCL   DS   F       Length of the routine name
RXFARGS   DS   A       Pointer to argument list
RXFRET    DS   A       Pointer to EVALBLOK for
*                      function RETURN result
```

On entry to the exit, the name of the invoked function is defined by the fields RXFFNC and RXFFNCL. The arguments to the function are pointed to by the field RXFARGS. The flag RXFFSUB is on if the invoked routine is by means of a CALL rather than as a function.

On return from the exit, values in RXIFFLAG indicate the status of the function processing. If neither RXFFERR nor RXFFNFND is on, then the routine has been successfully invoked and has run successfully. The field RXFRET may have the address of an EVALBLOK containing the returned result. If the routine is invoked as a function and a result is not returned, then the language processor returns error 44, `Function did not return data`. If the routine is invoked as a subroutine, then the returned result is optional.

The flag RXFFNFND in RXIFFLAG indicates that the exit could not locate the routine with the given name. The language processor returns error 43, `Routine not found`. The flag RXFFERR indicates that the parameters supplied to the routine are somehow incorrect. The language processor returns error 40, `Incorrect call to routine`.

The EVALBLOK containing the result is allocated by the exit and the storage is returned by the language processor. The maximum size for an EVALBLOK is 16Mb.

Note: The EXECCOMM interface is enabled during calls to the RXFNC exits.

RXCMD Process host commands.

RXCMDHST Invoke a host command.

```
RXIEXIT   DS   H      Exit code = 3
RXISUBFN  DS   H      Exit subfunction = 1
RXIUSER   DS   F      User word
RXICFLAG  DS   X      Exit processing control flags
RXIFFLAG  DS   X      Exit specific flags
RXCFFAIL  EQU  X'80'  Command FAILURE occurred
*                     (trappable via SIGNAL ON FAILURE)
RXCFERR   EQU  X'40'  Command ERROR occurred
*                     (trappable via SIGNAL ON ERROR)
RXIFEVAL  EQU  X'01'  Return code returned via EVALBLOK
RXIPLEN   DS   H      Length of plist in bytes
RXIRESRV  DS   F      Reserved for future use
RXCADDR   DS   CL8    Current ADDRESS setting
RXCCMD    DS   A      Pointer to the command
RXCCMDL   DS   F      Length of the command
RXCRETC   DS   A      Pointer to return code buffer
RXCRETCL  DS   F      Length of return code
```

On entry to the exit, the fields RXCRETC and RXCRETCL define a buffer that returns a value used for the return code in character format (i.e., numeric return codes must be formatted as a character string). The return code may have a nonnumeric value if desired. On return from the exit, RXCRETCL contains the length of the data placed in the buffer pointed to by RXCRETC.

If the buffer supplied is too small for the returning value, then the value is alternately returned using an EVALBLOK. In this case, the exit supplies an EVALBLOK and stores the address of the block in RXCRETC. The flag RXIFEVAL is then turned on to indicate that an EVALBLOK has been provided. If the value is returned by means of an EVALBLOK, the language processor handles releasing the EVALBLOK storage. The maximum size of an EVALBLOK is 16Mb.

The flags RXCFFAIL and RXCFERR are used by the exit to indicate that an ERROR or FAILURE condition has occurred. The definition of what constitutes an ERROR or FAILURE of a command is under the control of the exit. Under the default command processor, a negative return code is a FAILURE condition and a positive return code is an ERROR condition.

Note: The EXECCOMM interface is enabled during calls to the RXCMD exits.

RXMSQ Manipulate session queue.

This service supports a number of subfunctions. The subfunction request code is contained in the field RXISUBFN (found in RXISUBFN). The remainder of the parameter list depends on the particular subfunction invoked. The RXMSQ subfunctions and their parameter lists are:

RXMSQPLL Pull a line from the session queue.

```
RXIEXIT   DS   H       Exit code = 4
RXISUBFN  DS   H       Exit subfunction = 1
RXIUSER   DS   F       User word
RXICFLAG  DS   X       Exit processing control flags
RXIFFLAG  DS   X       Exit specific flags
RXMFEMPT  EQU  X'40'   Queue was empty
RXIFEVAL  EQU  X'01'   String returned via EVALBLOK
RXIPLEN   DS   H       Length of plist in bytes
RXIRESRV  DS   F       Reserved for future use
RXMRETC   DS   A       Pointer to return value buffer
RXMRETCL  DS   F       Length of return value
```

On entry to the exit, the fields RXMRETC and RXMRETCL define a buffer that returns a value for the line removed from the session queue. If the buffer supplied is too small, then the value is returned using EVALBLOK. In this case, the exit supplies an EVALBLOK and stores the address of the block in RXMRETC. The flag RXIFEVAL is then turned on to indicate that an EVALBLOK has been provided. If the value is returned by means of an EVALBLOK, the language processor handles releasing the EVALBLOK storage. The maximum size of an EVALBLOK is 16Mb.

Although the CMS and GCS program stacks are limited to 255 bytes, there is no such limitation on a session queue provided by the RXMSQ exits.

On return from the exit, the RXMFEMPT flag indicates that there is no data on the queue, and no data has been returned. The contents of the buffer should be ignored.

RXMSQPSH Place a line on the session queue.

```
RXIEXIT   DS   H       Exit code = 4
RXISUBFN  DS   H       Exit subfunction = 2
RXIUSER   DS   F       User word
RXICFLAG  DS   X       Exit processing control flags
RXIFFLAG  DS   X       Exit specific flags
RXMFLIFO  EQU  X'80'   Stack the line LIFO
RXIPLEN   DS   H       Length of plist in bytes
RXIRESRV  DS   F       Reserved for future use
RXMVAL    DS   A       Pointer to line to stack
RXMVALL   DS   F       Length of line to stack
```

The line placed on the queue is the result of evaluating the expression specified on a PUSH or QUEUE instruction. This string can be any length up to 16Mb. It is the responsibility of the exit to handle truncation of this string if the exit has a restriction on the maximum width of the queue. The stacking order is indicated by the flag RXMFLIFO.

RXMSQSIZ Return the number of lines in the session queue.

```
RXIEXIT    DS   H       Exit code = 4
RXISUBFN   DS   H       Exit subfunction = 3
RXIUSER    DS   F       User word
RXICFLAG   DS   X       Exit processing control flags
RXIFFLAG   DS   X       Exit specific flags
RXIPLEN    DS   H       Length of plist in bytes
RXIRESRV   DS   F       Reserved for future use
RXMQSIZE   DS   F       Number of lines in stack
```

On return from the exit, RXMQSIZE contains the size of the data queue as a 32-bit unsigned number.

RXSIO Session I/O.

Note: The EXTERNALS built-in function always returns a value of zero when the RXSIO exit has been specified.

This service supports a number of subfunctions. The subfunction request code is contained in the field RXISUBFN. The remainder of the parameter list depends on the particular subfunction invoked. The RXSIO subfunctions and their parameter lists are:

RXSIOSAY Write a line to the character input stream. Called for SAY instruction to display output.

```
RXIEXIT    DS   H       Exit code = 5
RXISUBFN   DS   H       Exit subfunction = 1
RXIUSER    DS   F       User word
RXICFLAG   DS   X       Exit processing control flags
RXIFFLAG   DS   X       Exit specific flags
RXIPLEN    DS   H       Length of plist in bytes
RXIRESRV   DS   F       Reserved for future use
RXSVAL     DS   A       Address of line to display
RXSVALL    DS   F       Length of line to display
```

The line displayed at the terminal is the result of evaluating the expression specified on a SAY instruction. This string can be any length up to the size of the terminal (either by default system processing or by a call to RXSIOTLL). It is the responsibility of the exit to handle truncation of this string if the string is too long.

RXSIOTRC TRACE output processing. Call to output TRACE results.

```
RXIEXIT    DS   H       Exit code = 5
RXISUBFN   DS   H       Exit subfunction = 2
RXIUSER    DS   F       User word
RXICFLAG   DS   X       Exit processing control flags
RXIFFLAG   DS   X       Exit specific flags
RXIPLEN    DS   H       Length of plist in bytes
RXIRESRV   DS   F       Reserved for future use
RXSVAL     DS   A       Address of line to display
RXSVALL    DS   F       Length of line to display
```

The line to be displayed at the terminal is the result of a traced line. This string may be any length up to the size of the terminal (as determined by default system processing or by a call to RXSIOTLL). It is the responsibility of the exit to handle truncation of this string if the string is too long.

RXSIOTRD Read from character input stream.

```
RXIEXIT   DS  H       Exit code = 5
RXISUBFN  DS  H       Exit subfunction = 3
RXIUSER   DS  F       User word
RXICFLAG  DS  X       Exit processing control flags
RXIFFLAG  DS  X       Exit specific flags
RXIFEVAL  EQU X'01'   String returned via EVALBLOK
RXIPLEN   DS  H       Length of plist in bytes
RXIRESRV  DS  F       Reserved for future use
RXSRETC   DS  A       Pointer to return value buffer
RXSRETCL  DS  F       Length of return value
```

On entry to the exit, the fields RXSRETC and RXSRETCL define a buffer that may be used to return a value to be used for the line read from the terminal. If the buffer supplied is too small for the value that needs to be returned, then the value may alternately be returned using an EVALBLOK. In this case, the exit supplies an EVALBLOK and stores the address of the block in RXSRETC. The flag RXIFEVAL is then turned on to indicate that an EVALBLOK has been provided. If the value is returned by means of an EVALBLOK, the language processor handles releasing the EVALBLOK storage. The maximum size of an EVALBLOK is 16Mb.

RXSIODTR Debug read from character input stream.

```
RXIEXIT   DS  H       Exit code = 5
RXISUBFN  DS  H       Exit subfunction = 4
RXIUSER   DS  F       User word
RXICFLAG  DS  X       Exit processing control flags
RXIFFLAG  DS  X       Exit specific flags
RXIPLEN   DS  H       Length of plist in bytes
RXIRESRV  DS  F       Reserved for future use
RXSRETC   DS  A       Pointer to return value buffer
RXSRETCL  DS  F       Length of return value
```

On entry to the exit, the fields RXSRETC and RXSRETCL define a buffer that returns a value for the line read from the terminal. If the buffer supplied is too small, then a return code of -1 is returned.

RXSIOTLL Return maximum line length in bytes.

```
RXIEXIT   DS  H       Exit code = 5
RXISUBFN  DS  H       Exit subfunction = 5
RXIUSER   DS  F       User word
RXICFLAG  DS  X       Exit processing control flags
RXIFFLAG  DS  X       Exit specific flags
RXIPLEN   DS  H       Length of plist in bytes
RXIRESRV  DS  F       Reserved for future use
RXSSIZE   DS  F       Size of terminal in bytes
```

On return from the exit, RXSSIZE contains the width of the terminal as a 32-bit unsigned number.

This value is used by the LINESIZE built in function; and is used to break up lines created by SAY and TRACE. The RXSIOSAY and RXSIOTRC functions must be capable of handling lines of this length.

RXMEM Memory services.

This service supports a number of subfunctions. The subfunction request code is contained in the field RXISUBFN. The remainder of the parameter list depends on the particular subfunction invoked. The RXMEM subfunctions and their parameter lists are:

RXMEMGET Allocate memory.

```
RXIEXIT   DS   H       Exit code = 6
RXISUBFN  DS   H       Exit subfunction = 1
RXIUSER   DS   F       User word
RXICFLAG  DS   X       Exit processing control flags
RXIFFLAG  DS   X       Exit specific flags
RXMFLO24  EQU  X'80'   Storage must be allocated below
*                      the 16Mb line.
RXIPLEN   DS   H       Length of plist in bytes
RXIRESRV  DS   F       Reserved for future use
RXMSSIZE  DS   F       Size of storage to be
*                      allocated (in doublewords)
RXMADDR   DS   A       Address of allocated storage
```

On entry to the exit, RXMSSIZE contains the size of the block of storage to be allocated. On exit, RXMADDR should contain the address of the allocated storage. Out-of-storage conditions are reflected by setting RXMADDR to zero on return from the exit. The flag RXMFLO24 indicates that the storage must be allocated below the 16Mb line.

RXMEMRET Deallocate memory.

```
RXIEXIT   DS   H       Exit code = 6
RXISUBFN  DS   H       Exit subfunction = 2
RXIUSER   DS   F       User word
RXICFLAG  DS   X       Exit processing control flags
RXIFFLAG  DS   X       Exit specific flags
RXIPLEN   DS   H       Length of plist in bytes
RXIRESRV  DS   F       Reserved for future use
RXMSSIZE  DS   F       Size of storage to be
*                      released (in doublewords)
RXMADDR   DS   A       Address of storage to be
*                      released
```

On entry, the fields RXMSSIZE and RXMADDR contain the length and address of the storage to be released.

Note: Because calls to external functions and other exits can result in the language processor obtaining blocks of storage that were not allocated by calls to the RXMEMGET exit, the RXMEMRET exit should be prepared to handle these conditions. If desired, a return code of 1 can be used to cause a block of storage to be released by normal system means.

Note: Because the services provided by the RXMEM exits cannot support releasing storage in increments other than those allocated, the language processor does not release partial storage if the RXMEM exit has been specified.

RXHLT Halt processing.

This service supports a number of subfunctions. The subfunction request code is contained in the field RXISUBFN. The remainder of the parameter list depends on the particular subfunction invoked. The RXHLT subfunctions and their parameter lists are:

RXHLTCLR Clear Halt indicator.

```
RXIEXIT   DS   H       Exit code = 7
RXISUBFN  DS   H       Exit subfunction = 1
RXIUSER   DS   F       User word
RXICFLAG  DS   X       Exit processing control flags
RXIFFLAG  DS   X       Exit specific flags
RXIPLEN   DS   H       Length of plist in bytes
RXIRESRV  DS   F       Reserved for future use
```

This exit has no inputs or outputs. It signals the exit handling HALT processing that the condition has been recognized and should be cleared.

RXHLTTST Test Halt indicator.

```
RXIEXIT   DS   H       Exit code = 7
RXISUBFN  DS   H       Exit subfunction = 2
RXIUSER   DS   F       User word
RXICFLAG  DS   X       Exit processing control flags
RXIFFLAG  DS   X       Exit specific flags
RXHFHALT  EQU  X'80'   HALT condition occurred
RXIPLEN   DS   H       Length of plist in bytes
RXIRESRV  DS   F       Reserved for future use
RXHSTR    DS   A       Pointer to EVALBLOK
*                      containing an optional
*                      HALT string
```

On return from this exit, RXHFHALT indicates whether a HALT condition has occurred. The exit can also return a string that is available as CONDITION('Description') for a CALL ON HALT or SIGNAL ON HALT condition trap. This string is returned by means of an EVALBLOK. In this case, the exit supplies an EVALBLOK and stores the address of the block in RXHSTR. If the value is returned via an EVALBLOK, the language processor handles releasing the EVALBLOK storage. The maximum size of an EVALBLOK is 16Mb.

RXTRC Trace services.

RXTRCTST Test external trace indicator.

```
RXIEXIT   DS   H       Exit code = 8
RXISUBFN  DS   H       Exit subfunction = 1
RXIUSER   DS   F       User word
RXICFLAG  DS   X       Exit processing control flags
RXIFFLAG  DS   X       Exit specific flags
RXTFTRAC  EQU  X'80'   External TRACE setting
RXIPLEN   DS   H       Length of plist in bytes
RXIRESRV  DS   F       Reserved for future use
```

On return from this exit, RXTFTRAC indicates whether an external trace condition has occurred.

RXINI Initialization processing.

RXINIEXT Perform external initialization.

```
RXIEXIT   DS   H      Exit code = 9
RXISUBFN  DS   H      Exit subfunction = 1
RXIUSER   DS   F      User word
RXICFLAG  DS   X      Exit processing control flags
RXIFFLAG  DS   X      Exit specific flags
RXIPLEN   DS   H      Length of plist in bytes
RXIRESRV  DS   F      Reserved for future use
```

This exit has no inputs or outputs. It is called before the first instruction of the program is interpreted. The EXECCOMM interface is enabled when this exit is called.

RXTER Termination processing.

RXTEREXT Perform External Termination.

```
RXIEXIT   DS   H      Exit code = 10
RXISUBFN  DS   H      Exit subfunction = 1
RXIUSER   DS   F      User word
RXICFLAG  DS   X      Exit processing control flags
RXIFFLAG  DS   X      Exit specific flags
RXIPLEN   DS   H      Length of plist in bytes
RXIRESRV  DS   F      Reserved for future use
```

This exit has no inputs or outputs. It is called after the last instruction of the program is interpreted. The EXECCOMM interface is enabled when this exit is called.

Usage Notes

1. At invocation of the interpreter by an application program, the exit vector could contain codes that are not defined or are reserved for future use. These will be ignored.

2. The exit vector could contain the same code more than once. When obtaining the storage for the REXX WORKBLOK, the first occurrence of the RXMEM exit will be used. The first occurrence of the REXMEM exit will also be used to release the storage for the WORKBLOK. In all other cases, including the RXMEM exit, the last occurrence of an exit code will be the one used.

3. Upon return from one of the system exits to the interpreter, the return code might be a non-zero return code other than the documented plus 1 or minus 1. All negative return codes will be treated the same as a minus 1 and all positive return codes will be treated the same as a plus 1.

4. The EXTERNALS() built-in function will always return a value of zero when the RXSIO exit has been specified.

5. The EXECCOMM interface may be used by the RXINI, RXTER, RXCMD, and RXFNC exits. Return code -1 will be given back by EXECCOMM when it is called from any other exit routine.

6. When invoked on GCS with the RXMEM exit specified, there are three circumstances where the RXMEM exit is not used to obtain and release storage:

   - The REXX interpreter has been invoked via SVC 202 from a problem state program with exits specified. Storage is obtained for a register save area, then the SYNCH macro is used to switch to problem state. This area is retained throughout the processing of the EXEC, but is released just before the interpreter returns control back to the SVC handler to return control to the calling program. This can be avoided by using GCSCALL REXX.

- The EXECCOMM routine has been invoked via SVC 203 from a problem state program with exits specified. Storage is obtained for a register save area, then the SYNCH macro is used to switch to problem state. This area is retained throughout the processing of the EXECCOMM request, but is released just before EXECCOMM returns back to the SVC handler to return control to the calling program. This can be avoided by using GCSCALL EXECCOMM.
- The RXMEM exit is being called to obtain storage for the main WORKBLOK. Temporary storage is obtained for the RXMEM parameter list, which is released once the WORKBLOK has been obtained. The same sequence is used when calling RXMEM to release the storage occupied by the WORKBLOK.

  Note: Remember that the FIRST occurrence of the RXMEM exit found in the exit list is used to obtain and release the storage for the WORKBLOK. The LAST occurrence of the RXMEM exit found in the exit list is used for all other storage management requests.

7. The preceding usage notes are designated as "Product Sensitive Interfaces". Unless otherwise noted, all other programming information documented in this publication is considered to be part of the "General Programming Interface".
8. The descriptions of some of the exits state that a result may be returned in an EVALBLOK "if the supplied buffer is too small". In fact, the result may be returned via an EVALBLOK regardless of the size of the supplied buffer. It MUST be returned via an EVALBLOK if the supplied buffer is too small, providing the routine ends with a zero return code.

GPI end

New GCS ABEND Codes

GCS ABEND Codes

ABEND Code	Reason Code	Module Name	Cause of Abend	User Response
FCA	108	CSIRSS	The SYSREAD function overran the I/O buffer. Register 11 contains the parameter list address.	Correct the RXMEM exit being used.
FCA	109	CSIRSS	The SYSREAD function overran the I/O buffer. Register 11 contains the parameter list address.	Correct the RXMEM exit being used.
FCA	666	CSIRSS	Unrecognized function code. Register 8 contains the address where the SVC 202 was issued to invoke CSIRSS. The 8 character string representing the function code has been put into register 5 and 6. Register 11 contains the parameter list address.	Do not invoke the CSIRSS module. It is not intended for your use.
FCA	950	CSIREX	GCSCALL REXX was used by a problem state program running in key zero. Register 13 points to the save area where the caller's registers were saved.	Either correct your environment or use CMDSI to invoke the language interpreter.
FCA	95C	CSIRSS	The specified function must be called by SVC 202. Register 11 contains the parameter list address.	Do not invoke the CSIRSS module. It is not intended for your use.
FCA	95F	CSIRSS	A VALIDATE failed for the requested function. Register 8 contains the address where the SVC 202 was issued to invoke CSIRSS. Register 5 contains the VALIDATE return code. Register 11 contains the parameter list address.	• Do not invoke the CSIRSS module, it is not intended for your use • Correct the program using the GCSCALL SETCOMM macro or correct the application program that is modifying REXX storage • Correct the exit routine(s).

ABEND Code	Reason Code	Module Name	Cause of Abend	User Response
FCA	BBA	CSIREX	GCSCALL SETCOMM was used with an incorrect WORKBLOK address. Register 10 contains the original input address value.	Correct the program using the GCSCALL SETCOMM macro.
FCA	BBB	CSIREX	GCSCALL SETCOMM was used with an address of zero and the task block contains an incorrect WORKBLOK address. Register 10 contains the address from the task block.	Correct the program using the GCSCALL SETCOMM macro.
FCA	BBC	CSIRSS	GCSCALL SETCOMM was used with an incorrect WORKBLOK chain. Register 3 contains the original input address value.	Correct the program using the GCSCALL SETCOMM macro or correct the application program that is modifying REXX storage.
FCA	BBE	CSIREX	An exit routine failed to restore the correct register contents.	Correct the exit routine.
FCA	CF0	CSIRSS	CSIRSS was invoked by a non-REXX module. Register 8 contains the address where the SVC 202 was issued to invoke CSIRSS. Register 11 contains the parameter list address.	Do not invoke the CSIRSS module. It is not intended for your use.
FCA	CF9	CSIRSS	CSIRSS was invoked by a non-REXX module. Register 8 contains the address where the SVC 202 was issued to invoke CSIRSS. Register 11 contains the parameter list address.	Do not invoke the CSIRSS module. It is not intended for your use.
FCB	950	CSIREX	GCSCALL EXECCOMM was used by a problem state program running in key zero. Register 13 points to the save area where the callers registers were saved.	Either correct your environment or use CMDSI to invoke the language interpreter.

Design Changes

Internal Product Information

This section is intended to help you understand the design changes for this enhancement. It contains internal product information, which is provided as additional guidance on planning. The information in this section must not be used for programming purposes.

Changed Modules (CMS)

DMSEXE
DMSEXI
DMSREX
DMSRFF

Changed Modules (GCS)

CSIFNC
CSINUC
CSIREX
CSIRFF

New Modules (GCS)

CSIRSS

Changed Modules (REXX)

IXXRCN
IXXREV
IXXRFN
IXXRIN
IXXRKA
IXXRTC
IXXRVA
IXXRXE

New Modules (REXX)

IXXRXF

Changed Macros (CMS)

REXEXT

New Macros (CMS)

DMSRXM

Changed Macros (GCS)

CSIREXE
CSISIE

Changed Macros (REXX)

IXXMINT

New Macros (REXX)

EVALBLOK

Changed Control Blocks (CMS)

FBLOCK

Changed Control Blocks (GCS)

FBLOCK CSISIE

Changed Execs (CMS)

DMSSP
CMSLOAD

Changed Execs (GCS)

CSISP
GCSLOAD

Changed HELP Files

CALL	HELPREXX
SIGNAL	HELPREXX

New HELP Files

CONDITIO	HELPREXX

Bibliography

The following tables list the program update books that have been issued since VM/SP Release 4 became available. Refer to the appropriate column for the books that you may need for your installation. You can order any of the publications through the System Library Subscription Service (SLSS).

Table 1. Recent Program Update Books

Title	Order Number	VM/SP R5	VM/SP R6	APAR
Enhancements to the LINERD and LINEWRT Macros	GC24-5313		X	VM33157
Support for United States Department of Defense C2 Security	SC24-5384	X		VM33580
Transparent Services Access Facility Virtual Telecommunications Access Method Line Driver	GC24-5392		X	VM33116
CMS Session Services Command Restructure	GC24-5482	X	X	VM35846 VM35847
Alternate-VSAM	GC24-5399		X	VM33162
Security and Integrity Enhancements	GC24-5312		X	VM33986
Support for VM/Directory Maintenance Licensed Program Release 4	GT00-3328	X	X	VM33536
Transparent Services Access Facility Full Buffer Trace Enhancement	GC24-5404	X	X	VM36594
ECF/PASF Coexistence with IBM Enhanced Connectivity Facilities for VM/System Product	GC24-5488	X	X	VM36495 VM37139

Table 2 (Page 1 of 2). Program Update Books. Information from the following books is included in the VM/SP Release 6 base books.

Title	Order Number	VM/SP R4	VM/SP R5	APAR
IBM 9370 Processors, 9332 and 9335 Direct Access Storage Devices, and 9347 Tape Drive	GC24-5315	X	X	VM26890 VM26898
DASD DUMP/Restore Streaming Support Improvements	GC24-5359	X	X	VM27035
GCS/VSAM Support for Local Shared Resources/Deferred Write	GC24-5360	X	X	VM27102 VM27116
SPOOL Enhancement Accommodation	GC24-5361		X	VM27764
Enhancements to the IBM Enhanced Connectivity Facilities for VM/System Product	GC24-5295		X	VM28510
IBM 3380 Direct Access Storage Models AJ4/BJ4 and AK4/BK4	GC24-5371	X	X	VM29096 VM29121 VM29123
IBM 3990 Storage Control Models 1 and 2 and IBM Direct Access Storage Direct Channel Attach Model CJ2	GC24-5372	X	X	VM29507
Terminal Usability Enhancements	GC24-5309		X	VM30316
Automatic Re-IPL Enhancement	GC24-5391		X	VM30314
Transparent Services Access Facility 9370 Local Area Network Subsystems	GC24-5363		X	VM29422 VM29450
9332 Direct Access Storage Device Enhancements	GC24-5403		X	VM31148
DIAGNOSE Code X'64' Enhancement	GC24-5311	X	X	VM31012
DIAGNOSE Code X'E4' Enhancement	GC24-5376		X	VM31011
VM/VTAM and NetView™ Enhancements	GC24-5310		X	VM30315 VM30387

Table 2 (Page 2 of 2). Program Update Books. Information from the following books is included in the VM/SP Release 6 base books.

Title	Order Number	VM/SP R4	VM/SP R5	APAR
Productivity Aids National Language Support Enhancement	GC24-5400		X	VM32507 VM32508
National Language Support File Naming Conventions Enhancements	GC24-5418		X	VM33119 VM33161 VM33407
Backward Macro Compatibility Enhancement	GC24-5423		X	VM34760

Table 3. Program Update Books. Information from the following books is included in the VM/SP Release 5 base books.

Title	Order Number	VM/SP R4	APAR
Support of IX/370 Enhancements	SC24-5280	X	VM22781
3380 Direct Access Storage Device Models AE4/BE4	SC24-5281	X	VM22795 VM22796 VM22820
Security Enhancements	SC24-5317	X	VM23495
CMS Vector Processing Support and TXTLIB Enhancement	SC24-5332	X	VM24666
Logical Device Host Limit Relief	SC24-5327	X	VM24773
Programmer's Guide to the Server-Requester Programming Interface for VM/SP	SC24-5291	X	VM25998
3480 Volume Serial Error Recording	SC24-5329	X	VM25737
CMS Console Facility	SC24-5333	X	VM25980
Support of Auto-Deactivation of Restricted Passwords	SC24-5335	X	VM26007
OS Simulation Standard Label Tape Processing Exits	GC24-5334	X	VM26302
Support of ASCII	GC24-5328	X	VM26903
Support of IBM 3422 Magnetic Tape Subsystem	GC24-5336	X	VM26428 VM26492
CP Extended Data Stream Support for VM/Pass-Through Facility Release 3	GC24-5354	X	VM27087

NetView is a trademark of the International Business Machines Corporation.

Virtual Machine/System Product
REXX Enhancements
Order No. GC24-5406-00

READER'S COMMENT FORM

Is there anything you especially like or dislike about this book? Feel free to comment on specific errors or omissions, accuracy, organization, or completeness of this book.

Note: Do not use this form to report system problems or to request copies of publications. Instead, contact your IBM representative or the IBM branch office serving you.

Would you like a reply? ___YES ___NO

Please print your name, company name, and address:

____

____

____

____

IBM Branch Office serving you: ______________________________

Thank you for your cooperation. You can either mail this form directly to us or give this form to an IBM representative who will forward it to us.

GC24-5406-00

Reader's Comment Form

CUT
OR
FOLD
ALONG
LINE

Fold and tape   Please Do Not Staple   Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL   PERMIT NO. 40   ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
DEPARTMENT G60
PO BOX 6
ENDICOTT NY 13760-9987

Fold and tape   Please Do Not Staple   Fold and tape

IBM ®

Program Number
5664-167

File Number
S370/4300-34

GC24-5406-00

Printed in U.S.A.