REXX User's Guide

BIM-GSS

Version 5 Release 3A

CSI
INTERNATIONAL

REXX User's Guide, Version 5 Release 3A
November 2006

Published by Connectivity Systems, Inc.
8120 State Route 138, Williamsport OH 43164
Phone: 800-795-4914 Fax: 740-986-6022

E-Mail: *help@e-vse.com*

Internet: *http://www.e-vse.com*

Contents

About This Guide

Purpose ATG–1
Organization ATG–1
GSS Publications ATG–2
    Related Publications ATG–2
Diagnostic Procedures ATG–2
    Collecting Diagnostic Data ATG–3
    Interpreting Diagnostic Data ATG–3
        Calling Technical Support ATG–4

Chapter 1: Using the REXX Editor and Compiler

REXX Overview 1–1
    What Is REXX? 1–1
    Using REXX 1–1
    SDL/SVA Requirements 1–1
    Running REXX Procedures from JCL 1–2
    Enabling REXX Procedures 1–2
Enabling FAQSAO 1–2
    Running FAQSAO 1–3
    Running FAQSVSPO 1–3
    Defining the Application ID 1–3
    Terminating FAQSAO 1–3
    Terminating BIM-FAQS/ASO 1–3
Initializing the FAQSAO REXX Processor 1–4
    Accessing the ASO IMOD Initialization Directory List 1–4
IMOD Configuration Screen 1–5
    Accessing the IMOD Configuration Screen 1–5
Editing REXX IMODs 1–6
    REXX IMOD File Directory List 1–6
    Accessing the REXX IMOD File Directory List 1–7
    Actions on the REXX IMOD File Directory List 1–8
REXX IMOD Editor Screen 1–9
    REXX IMOD Format 1–9
Editing on the REXX IMOD Editor Screen 1–10

Overtyping Data 1–10
Entering Commands in the Prefix Area 1–11
Prefix-Area Commands 1–11
    CC, DD, and MM Commands 1–12
    C#, CC, M#, and MM Commands 1–12
Entering Command-Line Commands 1–12
Basic Commands 1–13
    Add 1–13
    Backward or Up 1–13
    Bottom or TOP 1–14
    Case 1–14
    Change 1–15
    DELete 1–15
    Down or Next 1–16
    FILE 1–17
    SAVE 1–17
    Search 1–17
Advanced Commands 1–19
    = (Recall) 1–19
    DUPlicate 1–19
    FFile 1–20
    GET 1–20
    Input 1–21
    Overlay Column 1–21
    PDS 1–22
    Quit 1–22
    SSAVE 1–23
    TAB 1–23
Using PF Keys While Editing 1–24
    Overview of PF Keys 1–24
    Detailed Summary of PF Keys 1–25
Compiling the REXX IMOD 1–27
    Saving an IMOD 1–27
    Filing an IMOD 1–27
    Determining the Member/User Name 1–28
    Performing Error Processing 1–28

Chapter 2: REXX Language

Overview 2–1
    REXX Instructions 2–1

- REXX Functions 2–1
- User Functions 2–2

Using Online Help 2–3
- Direct Help 2–3
- Accessing Online Help from REXX IMOD Editor 2–4
- Accessing Additional Help from Help Screen 2–5
- BIM REXX Help Menu 2–6
- Function Help 2–7

REXX General Usage 2–8
- Comments 2–8
- Symbols 2–8
- Strings 2–8
- Binary Strings 2–9
- Hexadecimal Strings 2–9
- Expressions 2–9
- Assignments 2–9

REXX Operators 2–10
- Prefix Operators 2–10
- Boolean Operators 2–10
- Algebraic Operators 2–10
- Comparators 2–10

Variables 2–11
- Simple Variables 2–11
- Stem Variables 2–11
  - Stem Variable Assignments 2–12
- Global Variables 2–12
- Global Stem Variables 2–12

How Arguments Are Passed 2–13
- Default Argument Passing Method 2–13
  - If a Message Triggered an IMOD 2–13
  - If a Command Triggered an IMOD 2–14
  - If an SMSG Triggered an IMOD 2–14

Differences Between BIM REXX and IBM REXX 2–15
- INTERPRET 2–15
- Loop Control Variables 2–15
- Labels 2–15
- Signal 2–16
- MAX and MIN 2–16
- Symbols 2–16
- TRACEs 2–16
- C2D 2–17

Floating Point Numbers 2–17
UPPER 2–17
DATE 2–17
DUMPSTG 2–17
Performance Hints 2–18
Global Variables 2–18
Stem Variables 2–18
Assignments = vs == 2–18
Long Strings 2–18
SUBSTR 2–18
Numbers 2–18
Comments 2–19
Statements 2–19
Arithmetic Items 2–19

Chapter 3: REXX Instructions

ADDRESS environment <command> 3–1
ARG template 3–2
RESULT 3–2
CALL function <expression<,expression>...> 3–3
DO 3–3
DROP 3–5
EXIT <expression> 3–6
IF 3–6
ITERATE <symbol> 3–7
THEN 3–7
ELSE 3–8
LEAVE <symbol> 3–8
NOP 3–9
NUMERIC 3–9
OTHERWISE 3–9
PARSE 3–10
PROCEDURE <expose <(> variable list <)> > 3–11
PULL 3–11
PUSH 3–12
QUEUE 3–12
RETURN 3–12
SAY 3–13
SELECT 3–14
SIGNAL 3–14

TRACE All/Commands/Error/Fail/Intermediate/Labels/Normal/Off/ Results...3–15
UPPER...3–15
WHEN...3–15

Chapter 4: REXX ADDRESS Environments

ADDRESS AO...4–1
ADDRESS CARD...4–3
ADDRESS CICS...4–4
ADDRESS CONSOLE...4–5
ADDRESS DISK...4–5
ADDRESS EPIC...4–6
ADDRESS EVENT...4–8
ADDRESS EXPLORE...4–9
ADDRESS OUTPUT...4–13
ADDRESS PDATE...4–14
ADDRESS PDS...4–15
ADDRESS POWER...4–16
ADDRESS PROGRAM...4–17
ADDRESS SCHEDULE...4–18
ADDRESS SYS...4–19

Chapter 5: REXX Functions

ABBREV(pattern,string,length)...5–2
ABS(number)...5–3
ADDRESS( )...5–4
ARG (<n<,option>>)...5–5
ASOENV ( )...5–6
BITAND(string1,string2,pad)...5–7
BITOR(string1,string2,pad)...5–8
BITXOR(string1,string2,pad)...5–9
B2C(binary-string)...5–10
B2X(binary-string)...5–11
CENTER(string,length<,pad>)...5–12
CENTRE(string,length<,pad>)...5–13
CLUSTER(parms)...5–14
COMPARE(string1,string2,pad)...5–15
COPIES(string,n)...5–16
CP(cmd,ASIS)...5–16

CPUID( ) 5–18
C2D(string<,n>) 5–18
C2X(string) 5–19
DATATYPE(string<,type>) 5–19
DATE(<option<,date<,'B'>>>) 5–21
DELSTR(string,n<length>) 5–22
DELWORD(string,n,length) 5–23
DIGITS( ) 5–23
D2C(number<,n>) 5–24
D2X(number<,n>) 5–25
ERRORTEXT(n) 5–25
FIND(string,tgt) 5–26
FORM( ) 5–27
FORMAT(number,<integer>,<decimal>) 5–27
FUZZ( ) 5–29
GETVIS(pid) 5–29
INDEX(haystack,needle,start) 5–31
INSERT(new,target,<n>,<length>,<pad>) 5–32
JOBACCT(pid) 5–33
JOBACCT('CPU') / JOBACCT('PAG') 5–34
JOBNAME(pid) 5–35
JUSTIFY(string,length,pad) 5–36
LASTPOS(needle,haystack,start) 5–37
LEFT(string,length,pad) 5–38
LENGTH(string) 5–39
LIBR(function,parms) 5–40
LINESIZE( ) 5–40
LISTCAT(parms) 5–41
MAX(number,number,...) 5–41
MESSAGE(<pid>,<count>,<scan>,<start>) 5–42
MIN(number,number,...) 5–43
MSG(mid<,start>) 5–44
OVERLAY(new,tgt,<n>,<len>,<pad>) 5–45
PHASE(pid) 5–46
PIDLIST(<type>) 5–47
POS(needle,haystack,start) 5–48
POST(event) 5–49
POWER(<queue>,<jobname>,<class>) 5–49
PWRCMD(cmd) 5–51
QUEUED( ) 5–52
RANDOM(min,max,seed) 5–52

READCONS(<data>) 5–54
REPLID(pid) 5–55
REVERSE(string) 5–56
RIGHT(string,len,pad) 5–56
SESSION(args) 5–57
SIGN(number) 5–58
SOURCELINE(<n>) 5–58
SPACE(string,n,pad) 5–59
STATUS(pid) 5–60
STRIP(string,option,char) 5–61
SUBSTR(string,start<,end><pad>) 5–62
SUBWORD(string,n<,length>) 5–63
SYMBOL(symbol) 5–63
TIME(option) 5–64
TRANSLATE(string,tblo,tbli,pad) 5–65
TRUNC(number<,n>) 5–65
USERID( ) 5–66
VALUE(name<,newvalue>) 5–67
VERIFY(string,ref<,MatchNomatch>,<start>) 5–68
VSAM(function, parms) 5–69
VSSPACE(parms) 5–69
VTAM('cmd',<time>) 5–70
WAIT(sec) 5–71
WORD(string,n) 5–72
WORDINDEX(string,n) 5–73
WORDLENGTH(string,n) 5–73
WORD POS(string,target<,start) 5–74
WORDS(string) 5–75
XRANGE(start,end) 5–75
X2B(hstring) 5–76
X2C(hstring) 5–76
X2D(hstring,n) 5–77

Appendix A: Sample REXX IMODs

Sample IMODs A–1
    $ADDRESS A–1
    $ARG A–1
    $BEEPASO A–1
    $CICSREP A–1
    $CMSREP A–1

- $CONSOLE A–2
- $CP A–2
- $CPUUSE A–2
- $CYCLE A–2
- $DC A–2
- $EOJ A–2
- $GETVIS A–2
- $JOB A–3
- $JOBACCT A–3
- $JOBNAME A–3
- $JOBNREP A–3
- $JOBTIME A–3
- $LOG A–4
- $MESSAGE A–4
- $MSG A–4
- $NODESET A–4
- $PA A–4
- $PHASE A–5
- $POST A–5
- $POWGET A–5
- $PWRCMD A–5
- $QT A–5
- $READCON A–5
- $REPLID A–6
- $REPLY A–6
- $SCRIPT A–6
- $STATUS A–6
- $SUBMIT A–6
- $TO A–6
- $VTAM A–7

$ADDRESS Sample IMOD A–7
$ARG Sample IMOD A–8
$BEEPASO Sample IMOD A–9
$CICSREP Sample IMOD A–11
$CMSREP Sample IMOD A–12
$CONSOLE Sample IMOD A–14
$CP Sample IMOD A–14
$CPUUSE Sample IMOD A–15
$CYCLE Sample IMOD A–17
$DC Sample IMOD A–21
$EOJ Sample IMOD A–21

$GETVIS Sample IMOD A–22
$JOB Sample IMOD A–24
$JOBACCT Sample IMOD A–24
$JOBNAME Sample IMOD A–26
$JOBNREP Sample IMOD A–27
$JOBTIME Sample IMOD A–29
$LOG Sample IMOD A–29
$MESSAGE Sample IMOD A–29
$MSG Sample IMOD A–30
$NODESET Sample IMOD A–31
$PA Sample IMOD A–32
$PHASE Sample IMOD A–34
$POST Sample IMOD A–36
$POWGET Sample IMOD A–36
$PWRCMD Sample IMOD A–39
$QT Sample IMOD A–39
$READCON Sample IMOD A–40
$REPLID Sample IMOD A–40
$REPLY Sample IMOD A–40
$SCRIPT Sample IMOD A–41
$STATUS Sample IMOD A–45
$SUBMIT Sample IMOD A–46
$TO Sample IMOD A–47
$VTAM Sample IMOD A–47
$WAIT Sample IMOD A–48
$WAKEUP Sample IMOD A–48

Glossary : Basic Terms

Index

About This Guide

The About This Guide chapter describes the organization of the *GSS REXX User's Guide*, and provides a summary of each of the documents that complete the GSS documentation set.

Purpose

The *GSS REXX User's Guide* describes how to use the REXX editor and compiler and explains the REXX language, instructions, and functions. Sample IMODs are also included.

Organization

The following summarizes the outline of this manual.

Chapter 1, Using the REXX Editor and Compiler, provides basic information about REXX, explains how to enable the REXX processor, and discusses basic and advanced REXX commands.

Chapter 2, REXX Language, discusses REXX general usage, and its operators and variables. Also discussed is how to use the online help.

Chapter 3, REXX Instructions, explains each REXX instruction, including its purpose and an example.

Chapter 4, REXX Address Environments, discusses the environments in which REXX commands are processed, including syntax and operands for each command.

Chapter 5, REXX Functions, explains REXX functions. A sample program is included for each function.

Appendix A, Sample REXX IMODs, presents some examples of REXX IMODs.

GSS Publications

The GSS user library consists of the following documents.

Guide	Description
CPR User's Guide	Explains how to configure, initiate, and maintain BIM-CPR. It also explains how to create definitions and use BIM-CPR's online panels as well as how to issue print commands from BIM-FAQS/ASO.
Message Guide	Contains error messages that are generated by the GSS. The messages are grouped by program, and each message includes a reason and corrective action.
REXX User's Guide	Describes how to use the REXX editor and compiler. It also defines the REXX language, instructions, ADDRESS environments, and functions.
Installation and Utilities Guide	Describes the procedures for installing GSS in a VSE environment.

Related Publications

Other manuals may have the information you are looking for. Additional information about the REXX language can be found in the following IBM manuals:

- *TSO Extensions REXX User's Guide*
- *TSO Extensions REXX Reference*
- *VM System Product Interpreter User's Guide*
- *VM System Product Interpreter Help*

Additional information about the REXX language can also be found in M. F. Cowlishaw's *The REXX Language: A Practical Approach to Programming (Prentice-Hall)*. M. F. Cowlishaw is the author of the REXX language

Diagnostic Procedures

Refer to the table below for a summary of the procedures you should follow if you have a problem with a CSI software product. Each of these procedures is detailed on the following pages.

Step	Action
1	Categorize the problem and collect data. See “Collecting Diagnostic Data.”
2	Try to identify the problem. See “Interpreting Diagnostic Data.”
3	Collect diagnostic data and call support. See “Calling Technical Support.”
4	Work with Technical Support to solve the problem.

Collecting Diagnostic Data

In the table below, use the left column to categorize the problem your site has encountered. Then, follow the instructions in the corresponding right column to generate useful diagnostic data.

For	Be Sure to Check
Installation errors	All output produced by MSHP when the product was installed.
Screen errors	Copies of the screens in error. .

Interpreting Diagnostic Data

After collecting the specified diagnostic data, write down answers to the following questions:

- What was the sequence of events prior to the error condition?
- What circumstances existed when the problem occurred and what action was taken?
- Has this situation occurred before? What was different then?
- Did the problem occur after a particular PTF was applied or after a new release of the software was installed?
- Was a new release of the operating system installed recently?
- Has the hardware configuration (tape drives, disk drives, and so forth) changed?

From the answers to these questions and the diagnostic data, try to identify the cause and resolve the problem. If it is determined that the problem is a result of an error in a CSI software product, contact CSI Technical Support.

Calling Technical Support

CSI International provides support for all its products.

If you are in North America, call 800-795-4914. Outside North America, call your local CSI Software Agent. You can also reach CSI Technical Support online at help@e-vse.com.

Please have the following information ready before contacting CSI Technical Support:

- All the diagnostic information described in "Collecting Diagnostic Data." Product name, product code and release number.
- Product name and release number of any other software you suspect is involved.
- Release level and PUTLEVEL of the operating system.
- Your name, telephone number and extension (if any).

Your company name

.

Chapter 1
Using the REXX Editor and Compiler

This chapter introduces the REXX language and explains how to use the REXX editor and compiler.

REXX Overview

What Is REXX?

REXX is a programming language. BIM describes REXX as an OAL (Operations Automation Language). REXX provides a simple and structured environment for elevating your automated operations to new levels. As a result, you can interface with the system console, VM, POWER, VTAM, CICS, BIM-FAQS/PCS, BIM-EPIC, CA-EXPLORE for VSE, and other products.

Using REXX

The BIM implementation of REXX is used to create Intelligence Modules (IMODs) and to execute them automatically within ASO. You use the REXX editor to create and modify the IMODs.

Additionally, you can initiate and run REXX IMODs in many environments. For example, an IMOD can be run as a batch job using BIM$RXBA or PCSBAT. You can also run REXX as part of JCL in a library PROC member. BIM-FAQS/PCS and BIM-FAQS/ASO users can run REXX IMODs using the FAQSAO task. The FAQSAO task can run IMODs on behalf of other BIM products through:

- the Global Event Manager (GEM)
- an BIM-FAQS/PCS event or command
- SMSG to VSE, console commands, online screens, and console messages (with BIM-FAQS/ASO)
- HTTP (CGI) interface
- Buffer interface

SDL/SVA Requirements

The following phases need to be in the SVA before REXX can be used through FAQSAO:

```
$FAQS,SVA
$FAQSAO,SVA
```

These will automatically be placed into the SVA if GSFTL FAQS is run during IPL. For more information on GSFTL, see the BIM-FAQS/ASO *User's Guide*.

Running REXX Procedures from JCL

You can run a REXX program from a VSE library procedure by using // EXEC PROC=*procedure*, where *procedure* is a REXX program beginning with a comment in columns 2-80. The REXX program is compiled and executed at execute proc time. Any data that is pushed to the stack is executed as JCL after procedure termination. For proper execution, the JCL should be pushed in LIFO order. For example:

```
// EXEC LIBR
AC S=DCMLIB.330
CATALOG REXXSAMP.PROC REPLACE=Y
 /* rexx comment */
PUSH '/*'
PUSH 'INCLUDE $LISTDIR'
PUSH 'FORECAST' DATE('0')
PUSH '// EXEC PCSEVRP'
PUSH '// EXEC GSFAQS'
PUSH '// PAUSE'
PUSH '* THIS IS A TEST JCL'
/+
/*
```

Enabling REXX Procedures

To enable support for executing REXX procedures, you must execute the following JCL to place the JOB control hooks:

```
// EXEC BIM$UTIL
ENABLE REXXPROC
/*
```

Enabling FAQSAO

If you want to execute BIM-FAQS/ASO REXX IMODs through events, SMSGs, console commands, online commands, console messages, browser requests, and buffer requests, the FAQSAO task must be active in a partition. The FAQSAO task can run as a main task or as a subtask of a long-running job such as POWER, VTAM, CICS, or BIM-FAQS/PCS's JCLSCHED.

Running FAQSAO

The FAQSAO task is used to multitask IMODs for BIM-FAQS/ASO and BIM-FAQS/PCS. Other BIM products may schedule IMODs to run GEM. If you are running BIM-FAQS/PCS, FAQSAO is automatically attached as a subtask of the scheduler.

To run FAQSAO as a main task, use the following JCL statement:

```
// EXEC FAQSAO,SIZE=FAQSAO
```

If BIM$TIDR, FAQSVMX, or FAQSIUX is running, Don't run FAQSAO in the same partition, if possible. Use BIM$UTTS to subtask FAQSAO , if necessary.

Running FAQSVSPO

FAQSVSPO is a VTAM Secondary Programmable Operator used to issue VTAM commands and retrieve the results in a REXX IMOD for processing. FAQSAO and FAQSVSPO must run in the same partition, but they don’t have to run under VTAM. The following JCL enables FAQSAO and FAQSVSPO using BIM$UTTS. For more information on BIM$UTTS, see the *GSS Installation and Utilities Guide*.

```
// EXEC BIM$UTTS,PARM='FAQSAO#FAQSVSPO'
```

Defining the Application ID

You must add the application ID for the FAQSVSPO task to run. Add the following statement to your VBUILD command list:

```
FAQSVSPO APPL AUTH=(SPO)
```

Terminating FAQSAO

To terminate the FAQSAO task, use the AO SHUTDOWN command.

Terminating BIM-FAQS/ASO

To terminate BIM-FAQS/ASO, use the following procedure.

1. Execute the AO SHUTDOWN command to terminate the FAQSAO task.

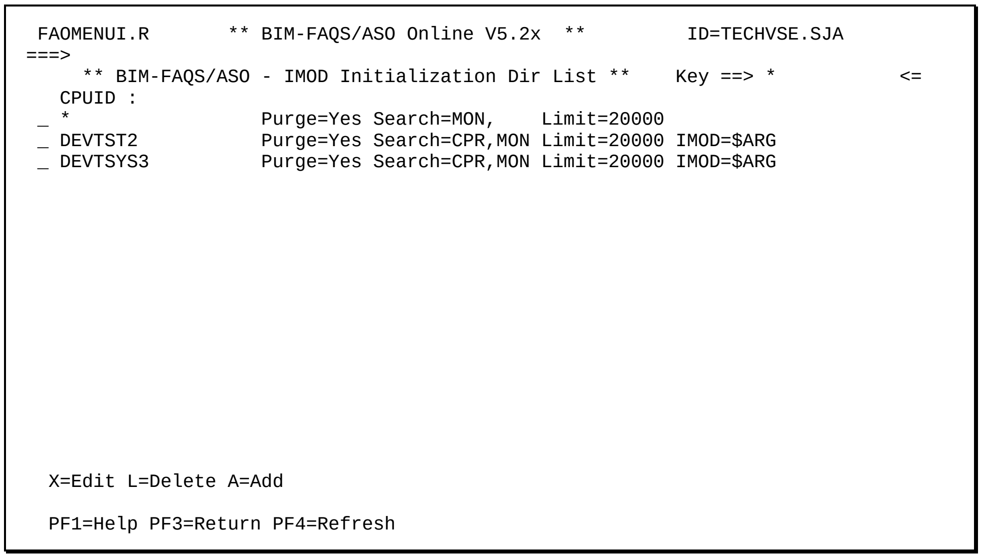
2. Use the GSFAQS DISABLE AO command to terminate the SMSG and AR hooks.

Initializing the FAQSAO REXX Processor

When the FAQSAO task is initialized, the AOINIT IMOD is executed. AOINIT looks for initialization and configuration data for a particular CPU ID or a default file of *. Initialization and configuration data is defined on the ASO IMOD Initialization Directory List.

Accessing the ASO IMOD Initialization Directory List

From the BIM-FAQS/ASO Initialization and Configuration Menu, select the REXX IMOD Initialization and Tailoring option (option R) to display the ASO IMOD Initialization Directory List.



```
 FAOMENUI.R       ** BIM-FAQS/ASO Online V5.2x  **         ID=TECHVSE.SJA
===>
     ** BIM-FAQS/ASO - IMOD Initialization Dir List **    Key ==> *           <=
   CPUID :
 _ *                 Purge=Yes Search=MON,    Limit=20000
 _ DEVTST2           Purge=Yes Search=CPR,MON Limit=20000 IMOD=$ARG
 _ DEVTSYS3          Purge=Yes Search=CPR,MON Limit=20000 IMOD=$ARG










  X=Edit L=Delete A=Add

  PF1=Help PF3=Return PF4=Refresh
```

Sample ASO IMOD Initialization Directory List

The Action and PF-Key Functions are discussed next.

Action/PF Key	Function
A	Add a REXX IMOD Initialization
L	Delete a defined REXX IMOD Initialization
X	Edit a defined REXX IMOD Initialization
PF1	Access help information for this screen
PF3	Return to the previous screen
PF4	Refresh the current display

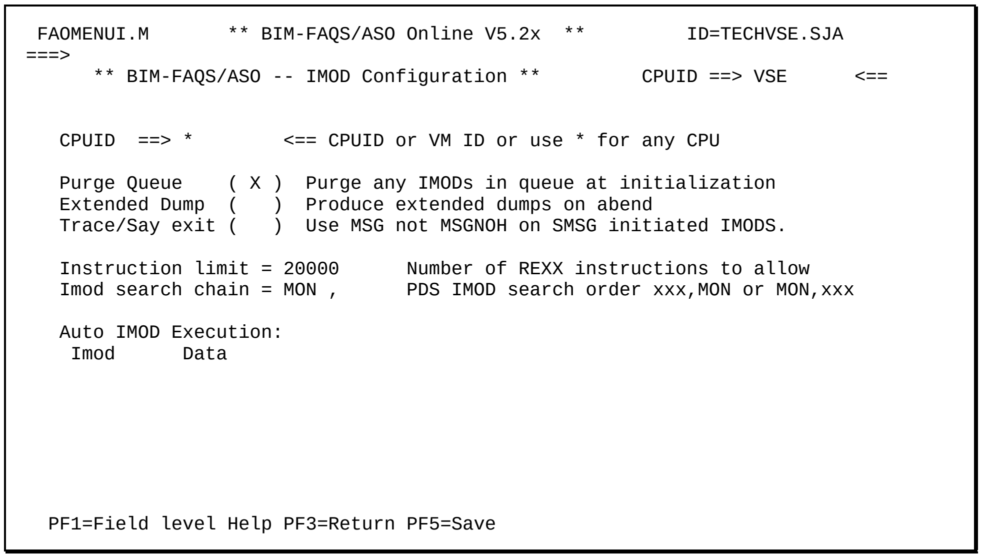
IMOD Configuration Screen

Accessing the IMOD Configuration Screen

From the ASO IMOD Initialization Directory List, use the action codes to access the IMOD Configuration Screen.

For example, to edit a specific initialized CPU ID, enter **X** in the input field next to the desired CPU ID to display the IMOD Configuration screen for that CPU. You can alter information by typing over existing information; then press PF5 to save.

To add a new initialized CPU ID, enter **A** in the input field next to any CPU ID. A blank IMOD Configuration screen is displayed, shown next.



```
 FAOMENUI.M       ** BIM-FAQS/ASO Online V5.2x  **         ID=TECHVSE.SJA
===>
      ** BIM-FAQS/ASO -- IMOD Configuration **         CPUID ==> VSE      <==


   CPUID  ==> *        <== CPUID or VM ID or use * for any CPU

   Purge Queue    ( X )  Purge any IMODs in queue at initialization
   Extended Dump  (   )  Produce extended dumps on abend
   Trace/Say exit (   )  Use MSG not MSGNOH on SMSG initiated IMODS.

   Instruction limit = 20000      Number of REXX instructions to allow
   Imod search chain = MON ,      PDS IMOD search order xxx,MON or MON,xxx

   Auto IMOD Execution:
    Imod      Data







  PF1=Field level Help PF3=Return PF5=Save
```

Sample IMOD Configuration Screen

The fields are discussed next.

CPUID	The CPU ID or VM machine name where the FAQSAO task is initiated. The CPU ID '*' allows the file to be loaded on any CPU if a matching CPUID or VM machine name is not found. The CPU ID can be modified on the screen to enable you to copy an entry to a new or existing file.
Purge	Indicates whether or not AOINIT purges outstanding IMODs queued for execution when FAQSAO is initialized.
Ext Dmp	Indicates whether or not extended dumps are on when FAQSAO is initialized. Ext Dmp should be off unless requested by BIM Technical Support.

Limit	Indicates the number of REXX instructions to allow in an IMOD execution. Setting a limit enables you to prevent loops that can be coded in an IMOD. When the limit is reached, the IMOD is canceled. If this field is set to *, infinite loops can be coded.
Imod search chain	Specifies PDSs to search for IMODs. A maximum of two PDSs can be specified as search targets. *MON* is the default. BIM-FAQS/ASO searches the specified PDSs in the order in which they appear in this search chain. For example, specifying *Imod search chain=CPR, MON* means that the SYS$CPR PDS is searched first, then the SYS$MON PDS.
IMOD	Indicates one or two IMODs to initialize. You can pass optional data to the IMODs.

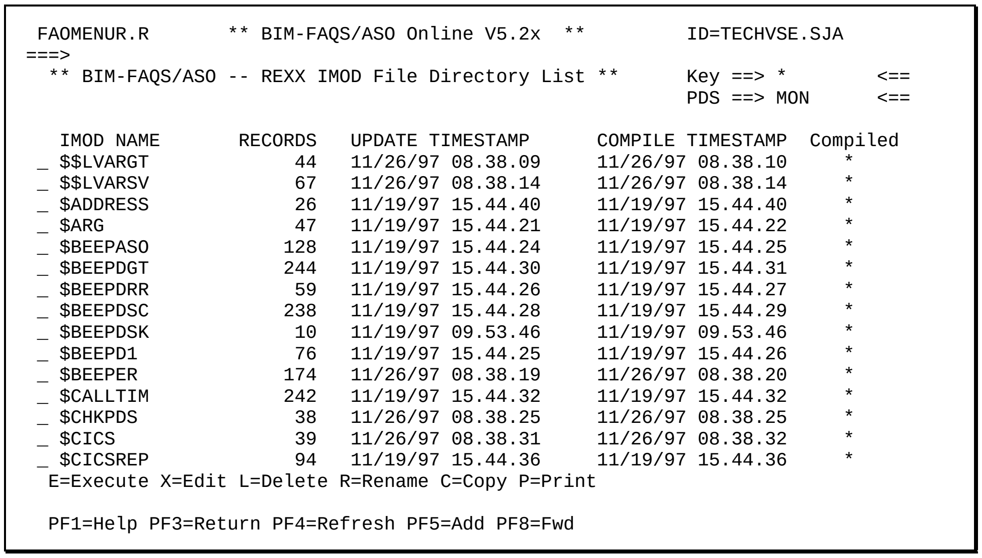
Editing REXX IMODs

REXX IMOD File Directory List

Starting from the REXX IMOD File Directory List, you can create REXX IMODs using the REXX language. For details on the REXX language, see Chapter 2, "REXX Language."

Accessing the REXX IMOD File Directory List

Select the REXX IMOD Directory option from either the BIM-FAQS/ASO Main Menu or the BIM-FAQS/PCS Main Menu, and the REXX IMOD File Directory List is displayed.



```
 FAOMENUR.R       ** BIM-FAQS/ASO Online V5.2x  **         ID=TECHVSE.SJA
===>
  ** BIM-FAQS/ASO -- REXX IMOD File Directory List **      Key ==> *        <==
                                                           PDS ==> MON      <==

   IMOD NAME       RECORDS   UPDATE TIMESTAMP      COMPILE TIMESTAMP  Compiled
 _ $$LVARGT             44   11/26/97 08.38.09     11/26/97 08.38.10     *
 _ $$LVARSV             67   11/26/97 08.38.14     11/26/97 08.38.14     *
 _ $ADDRESS             26   11/19/97 15.44.40     11/19/97 15.44.40     *
 _ $ARG                 47   11/19/97 15.44.21     11/19/97 15.44.22     *
 _ $BEEPASO            128   11/19/97 15.44.24     11/19/97 15.44.25     *
 _ $BEEPDGT            244   11/19/97 15.44.30     11/19/97 15.44.31     *
 _ $BEEPDRR             59   11/19/97 15.44.26     11/19/97 15.44.27     *
 _ $BEEPDSC            238   11/19/97 15.44.28     11/19/97 15.44.29     *
 _ $BEEPDSK             10   11/19/97 09.53.46     11/19/97 09.53.46     *
 _ $BEEPD1              76   11/19/97 15.44.25     11/19/97 15.44.26     *
 _ $BEEPER             174   11/26/97 08.38.19     11/26/97 08.38.20     *
 _ $CALLTIM            242   11/19/97 15.44.32     11/19/97 15.44.32     *
 _ $CHKPDS              38   11/26/97 08.38.25     11/26/97 08.38.25     *
 _ $CICS                39   11/26/97 08.38.31     11/26/97 08.38.32     *
 _ $CICSREP             94   11/19/97 15.44.36     11/19/97 15.44.36     *
  E=Execute X=Edit L=Delete R=Rename C=Copy P=Print

  PF1=Help PF3=Return PF4=Refresh PF5=Add PF8=Fwd
```

Sample ASO IMOD Initialization Directory List

The REXX IMOD File Directory List Fields are discussed next.

Key ==> <==	Criteria to display members. * alone displays all members. * as a wildcard replaces one or more characters in a member name. ? as a wildcard replaces one character.
PDS==>	PDS to search for REXX IMODs. If you have accessed this screen from the Console Command Definition panel or the Console Action Definition panel, you may need to change this value to reflect the actual location of the PDS you want to edit or execute.

_ (input field)	Action to perform against the IMOD:

Edit	Edits the IMOD
Delete	Deletes the IMOD
Rename	Renames the IMOD
Copy	Copies an IMOD. Use this command to add IMODs by modifying the newly created IMOD.
Print	Submits the IMOD for a batch print job.

IMOD NAME	Name of the IMOD. An IMOD is an extended command that is written to respond to system actions and to execute specific actions.
RECORDS	Number of lines in the IMOD.
UPDATETIMESTAMP	Date and time the member was last updated.
COMPILE TIMESTAMP	Date and time the member was last compiled.
Compiled	Whether or not the IMOD is compiled. An * indicates it is compiled.

Here are the REXX IMOD File Directory List PF Keys:

PF Key	Function
PF1	Displays online help for this screen
PF3	Returns to BIM-FAQS/ASO Main Menu
PF4	Refreshes the list when you add an IMOD
PF5	Adds a new REXX IMOD
PF8	Displays the next screen of the list
PF7	Displays the previous screen of the list

Actions on the REXX IMOD File Directory List

You can use the REXX IMOD File Directory List to delete, add, edit, or execute a REXX IMOD.

To execute an IMOD, type **E** next to the IMOD name and press Enter.

To delete an IMOD, type **L** next to the IMOD name and press Enter.

To add an IMOD, press PF5 (Add) to display the FAQS/ASO IMOD ADD screen. Enter the IMOD name you want to add and press Enter.

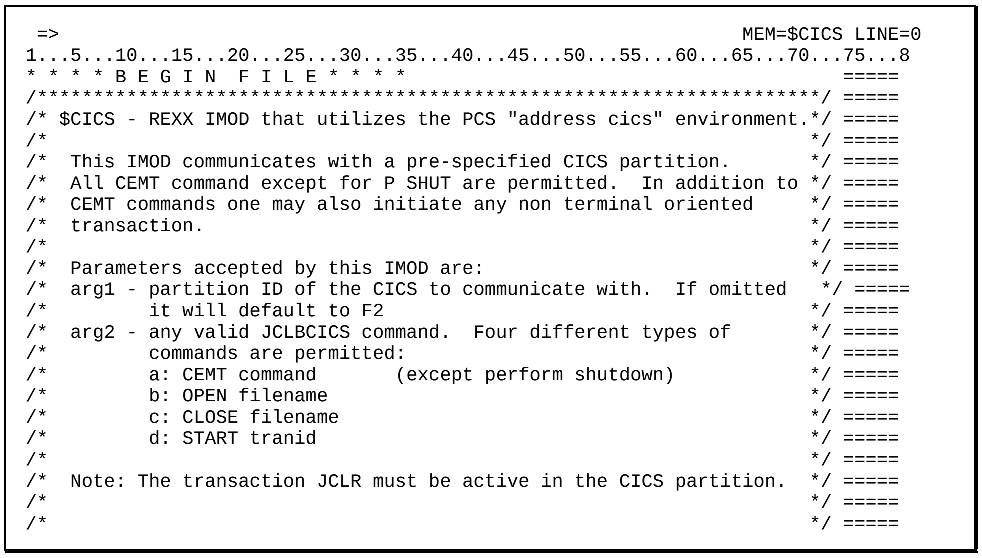
To select an IMOD for editing, type **X** in front of the IMOD name, or just place the cursor next to the name and press Enter. This takes you to the editor (REXX IMOD Editor screen).

REXX IMOD Editor Screen

The REXX IMOD Editor screen is displayed after you have selected the name of an IMOD to edit from the REXX IMOD File Directory List.

As shown on this screen, the display of an existing IMOD consists of:

- A command line at the top of the display (=>)
- 72-character lines used to define the particular IMOD
- A prefix area (=====)



```
 =>                                                              MEM=$CICS LINE=0
1...5...10...15...20...25...30...35...40...45...50...55...60...65...70...75...8
* * * * B E G I N  F I L E * * * *                                          =====
/**********************************************************************/ =====
/* $CICS - REXX IMOD that utilizes the PCS "address cics" environment.*/ =====
/*                                                                    */ =====
/*  This IMOD communicates with a pre-specified CICS partition.       */ =====
/*  All CEMT command except for P SHUT are permitted.  In addition to */ =====
/*  CEMT commands one may also initiate any non terminal oriented     */ =====
/*  transaction.                                                      */ =====
/*                                                                    */ =====
/*  Parameters accepted by this IMOD are:                             */ =====
/*  arg1 - partition ID of the CICS to communicate with.  If omitted   */ =====
/*         it will default to F2                                      */ =====
/*  arg2 - any valid JCLBCICS command.  Four different types of       */ =====
/*         commands are permitted:                                    */ =====
/*         a: CEMT command       (except perform shutdown)            */ =====
/*         b: OPEN filename                                           */ =====
/*         c: CLOSE filename                                          */ =====
/*         d: START tranid                                            */ =====
/*                                                                    */ =====
/*  Note: The transaction JCLR must be active in the CICS partition.  */ =====
/*                                                                    */ =====
/*                                                                    */ =====
```

Sample REXX IMOD Editor Screen

REXX IMOD Format

The first line of a REXX IMOD must begin with a comment line to identify the program as a REXX IMOD. A comment begins with /* and ends with */.

The sample REXX IMOD Editor screen above uses the first several comment lines to describe the IMOD and the remaining lines to define REXX commands.

Columns 73-80 (Prefix Area)

Columns 73-80 are the prefix area of the REXX IMOD Editor Screen. The prefix area can display the following.

Prefix area	Five equal signs (=====). This is the default. You can enter commands in the prefix area when it displays equal signs.
Relative-sequence number	A relative-sequence number beginning with 1. The relative-sequence number can be used in the GET command to designate only a portion of the IMOD member. You can enter commands in the prefix area when it displays relative-sequence numbers.
User record data	Column 80 of user data is never displayed. It is used for the screen-attribute character.
Blank	You cannot enter commands in the prefix area when it is blank.

PF2 toggles through the three display options: =====, relative-sequence numbers, and blank.

Editing on the REXX IMOD Editor Screen

There are four ways to edit a REXX IMOD member:

- Overtype existing data lines
- Enter prefix-area commands
- Enter command-line commands
- Use PF keys

Overtyping Data

You can overtype existing data to add command lines. The Input command enters a full-screen entry mode. You stay in input mode until you press Enter without any alterations.

By default, columns 1-72 of the member are displayed as data lines on the display. If the prefix area is not used, columns 1-79 are displayed. Column 80 cannot be changed because it is used for the screen-attribute character.

On a display, all visible positions can be altered and overtyped. Positions not visible cannot be changed unless a tab forces the entered data into the invisible area.

You can also place the cursor on any character on the screen and use the 3270 character DELETE, INSERT, and REPLACE features.

Entering Commands in the Prefix Area

The area on the right of the REXX IMOD Editor screen (columns 73-80) displays either equal signs or the relative-sequence number of the line. This area is the prefix area. The prefix area provides a convenient means of entering line-editing commands. From the prefix area, you can enter as many commands as you have lines of data showing before pressing Enter.

If relative-sequence numbers are displayed instead of equal signs (=====), commands must be left-justified and preceded by a number identifier. You can overtype the relative-sequence numbers.

Commands entered in the prefix area can be prefixed or suffixed with a number to affect multiple lines.

Prefix-Area Commands

To edit in the prefix area, you can use the following prefix-area commands:

Command	**Function**
A#	Adds # blank lines after this current line.
C#	Copies # lines, starting with this current line.
D#	Deletes # lines, starting with this current line.
F	Moves/Copies lines, starting with the line after this current line.
M#	Moves # lines, starting with the current line.

P	Moves/Copies lines, starting with the line before this current line.
/	Makes this line the TOP line.
“	Duplicates this current line.
CC	Defines first or last lines of block to copy.
DD	Defines first or last lines of block to delete.
MM	Defines first or last lines of block to move.

CC, DD, and MM Commands

The CC, DD, and MM commands apply to all lines delimited by the two lines that contain the command. For example, if lines four and eight contain DD in their command areas, lines four through eight are deleted.

C#, CC, M#, and MM Commands

The C#, CC, M#, and MM commands require a command that indicates where to move or copy the lines. The command is either P or F. P moves or copies the defined lines before the designated line. F moves or copies the defined lines after the designated line.

Entering Command-Line Commands

The command line is the first line of the display. Commands entered on the command line are more flexible than prefix-area commands.

There are two types of command-line commands: basic commands (used for screen editing) and advanced commands (used for editing and adding IMOD members). Both types of command-line commands are described in the following sections.

The help screen (accessed by entering HELP) lists the basic command-line commands.

Basic Commands

Add

Purpose

Adds blank lines in a member.

Syntax

```
Add# or A# or #Add or #A
```

# represents the relative number of lines to add. If a number is omitted, one is assumed.

Usage Notes

The command entered in the command line alone adds the blank lines after the first line shown on the screen.

Example

A2 or 2A entered on the command line adds two lines to the member. For example, if a / is placed in line 31's prefix area, two blank lines are added on lines 32 and 33.

Backward or Up

Purpose

Moves backward in a member one or more lines.

Syntax

```
U# or #U
```

# represents the relative number of lines to move. If a number is omitted, one (1) is assumed.

Usage Notes

The screen moves backward the specified number of lines from the current top line.

Example

U2 and 2U move backward two lines. U20 is the same as pressing PF7 on a Mod 2 terminal.

Bottom or TOP

Purpose

The Bottom command displays the END FILE line as the first (and only) line on the screen. The TOP command displays the first line of a member as the first screen line.

Syntax

```
BOT/TOP
```

Case

Purpose

Flips the current case setting. The editor is normally in uppercase/lowercase mode, and does not translate what is entered.

Syntax

```
CASE
```

Usage Notes

If you enter the CASE command, the editor translates all lowercase to uppercase. Enter the command again, and the lowercase characters remain the same.

This command is useful when entering all uppercase data, such as programs, on a terminal that supports both uppercase and lowercase.

Change

Purpose

Locates and modifies specific text in a member.

Syntax

```
C/search_string/replacement_string/**
```

Slashes delimit the command, the search string, the replacement string, and the asterisks are optional.

If no asterisks are specified, then only the first occurrence starting at the current line or the last change is performed.

If a single asterisk is specified, the first occurrence on each line is changed, starting at the current line or the last change that was performed.

If double asterisks are used, then all occurrences of the search string are changed, beginning with the current line or the last change that was performed.

Usage Notes

The search and replacement strings do not have to be the same length. The replacement string can be null.

If you omit the asterisks, only the first occurrence of the search string is changed. The top line of the screen becomes the line on which the search string is found. The first occurrence of the search string is displayed on the top line.

Example

C/3380/3390/* changes the first occurrence of 3380 to 3390 on each line.

DELete

Purpose

Deletes one or more lines from a member.

Syntax

```
DEL# or #DEL
```

# represents the relative number of lines to delete. If a number is omitted, one (1) line is assumed.

Usage Notes

Lines are deleted, beginning with the first line shown on the screen or with the line pointed to by a slash (/) in the prefix area.

If the number of lines supplied is more than the number of lines remaining in the file, all remaining lines are deleted.

Example

DEL3 deletes three lines, starting with the current line.

Down or Next

Purpose

Moves forward in a member one or more lines.

Syntax

```
D# or N# or #D or #N
```

D and N synonymous. # represents the relative number of lines to move. If the # is omitted, one (1) line is assumed.

Example

7N moves the screen forward seven lines. 20N is the same as pressing PF8 on a Mod 2 terminal.

FILE

Purpose

Accomplishes the following:

- Saves the file to the member shown after MEM=.
- Compiles the file, creating an executable IMOD.
- Exits the REXX editor

Syntax

```
FILE <member_name>
```

Usage Notes

Optionally, specify a member name to save the file to a new member. Use FFILE to save the file to an existing member other than the one shown after MEM=.

SAVE

Purpose

Saves the file to the member shown after MEM=.

Syntax

```
SAVE <member_name>
```

Usage Notes

Optionally, specify a member name to save the file to a new member. Use SSAVE to save the file to an existing member other than the one shown after MEM=.

Search

Purpose

Locates specific search strings.

Syntax

```
S/search_string/
```

Usage Notes

If the data specified in the search string is located, the line where it is found is made the first line on the screen and the cursor is placed at the beginning of the line. If the search string is not found, the screen remains unchanged and a message displays. The CASE setting determines whether the scan is uppercase only, or uppercase and lowercase.

Example

S/3330/ gets the first occurrence of 3330. You can also enter /3330/ or /3330.

Advanced Commands

= (Recall)

Purpose

Recalls and executes the last-executed command.

Syntax

```
=
```

Usage Notes

The recalled command is not displayed on the command line, and the command line is cleared for the next screen.

DUPlicate

Purpose

Duplicates lines on a screen.

Syntax

```
DUP
```

Usage Notes

A slash in the prefix area can be used to mark the line to duplicate.

FFile

Purpose

FFILE accomplishes the following: saves the file to an existing member other than the one shown after MEM=, compiles the file creating an executable IMOD, and exits the REXX editor.

Syntax

```
FFILE <member_name>
```

Usage Notes

The file is saved to either the member you specify with the command, or the member specified with the last SAVE or SSAVE command.

GET

Purpose

Inserts data from another member.

Syntax

```
GET member_name<,start,count>
```

Examples

GET TEMP inserts an entire member named TEMP.

GET PAUSE,2,4 inserts lines 2-6 from the PAUSE member.

Usage Notes

The specified member can be another member or a member of other source library books.

The start and count values are optional. If present, they specify the first line to insert and the number of lines to insert. Data is inserted after the current line or, if the cursor is positioned in the data area, after the cursor line.

After using the GET command, you must supply a member name for any of the save or file functions.

Input

Purpose

Like the ADD command, inserts new lines into a member. Input places the editor in full-screen entry mode.

Syntax

```
Input
```

Usage Notes

The editor displays the cursor at the top of the screen and fills the remainder of the screen with blank lines. Add lines by typing over the blank lines with new data.

Each time you press Enter, the editor displays the last line entered at the top of the screen and again fills the screen with blank lines. To restore the normal display, press Enter. You can also press Enter one last time without altering any data on the screen to exit insert mode.

Overlay Column

Purpose

Replaces one or more characters in a statement.

Syntax

OC*nn cccc*

nn is the desired column number (1-80) and *cccc* are the characters to store on top of that column.

Usage Notes

Position the cursor on the desired statement and press Enter. Overlay column is the only way to alter column 80 of a statement.

PDS

Purpose

Selects a specific PDS to be accessed as the current library. The default PDS for IMOD storage is PDS=MON.

Syntax

```
PDS=xxx
```

xxx is the filename of the PDSVSE dataset.

PDS=CUA$*nnn*

nnn is the device address to which the CMS minidisk is linked.

Usage Notes

The filename must be defined in either a standard label job execution, or in the JCL used to initiate the partition where the editor is running for the PDS=xxx dataset.

For CMS minidisk access, the CMS minidisk must be linked to a VSE disk with the same device characteristics, and the device must also be DVCUPed by VSE.

Quit

Purpose

Exits the editor without saving the member being edited.

Syntax

```
Quit
```

Usage Notes

If the member has been altered, a message displays, and you have to type QUIT again to force exit.

To exit the editor immediately without this message, enter **QQ**.

SSAVE

Purpose

Saves the file to an existing member other than the one shown after MEM=.

Syntax

```
SSAVE <member_name>
```

Usage Notes

The file is saved to either the member you specify or the member specified with the last SAVE or SSAVE command.

TAB

Purpose

Sets up to 10 tabs on a screen. A T displays wherever a tab is set.

Syntax

```
TAB t,nn,nn,nn,...
```

Tab character *t* can be any character. Tab settings *nn* must be entered in ascending numerical sequence.

Usage Notes

At least one tab position must be entered. To clear the tab settings, enter **TAB CLEAR**.

Using PF Keys While Editing

PF keys allow you to issue certain commands while using the REXX IMOD function. PF-key values described next are the system defaults.

To view the current PF-key settings for REXX IMOD displays, enter HELP while editing (X=Edit) a REXX IMOD.

Overview of PF Keys

The PF-key functions listed next are available by pressing the designated key or by entering PF and the key number. If you enter PF11 NEWNAME on the command line, the current member is stored in the default library as NEWNAME.

PF Key	Function
PF1/13	Help
PF2/14	Toggle prefix area on/off
PF3/15	Exit
PF4/16	Unused
PF5/17	Unused
PF6/18	Recall previous command
PF7/19	Scroll back page
PF8/20	Scroll forward page
PF9/21	Repeat previous command
PF10/22	Unused
PF11/23	Save file (make permanent update)
PF12/24	Submit file as jobstream

Detailed Summary of PF Keys

The following table describes the defined PF-key settings in more detail. If you have 24 PF keys, keys 13-24 correspond exactly to keys 1-12.

PF Key	Function
PF1	Displays the Help screen. Press Enter to return to the editor.
PF2	Turns the prefix area on and off. The area to the right of the screen that normally contains five equal signs (=====) is called the prefix area. The prefix area can be all equal signs (the default) or the relative line number of the member being viewed. It can also be turned off. When the prefix area is turned off, positions 73-79 of the member are displayed. Each time you press PF2, the prefix area changes from equal signs, to numbers, and then to data. When you use the relative-sequence number format, you must enter commands left-justified, preceded by the repeat count (if any).
PF3	Exit the editor. If the member has been updated and not saved, a message warns you that the member has been changed. To exit anyway, press PF3 again.
PF6	Displays the previous command. The editor stacks the last six commands entered on the command line. PF6 recalls them one at a time in round-robin fashion. The recalled command appears on the command line. Press Enter to execute the command again.
PF7	Scrolls back a page (20 lines through a member on a Mod 2 terminal). The first two lines on the screen become the bottom two of the next screen.
PF8	Scrolls forward a page (20 lines through a member on a Mod 2 terminal). The last two lines on the screen become the top two of the next screen.
PF9	Repeats the previous command.

Continued

Continued

PF Key	Function
PF11	Saves the member in the PDS library. If no member name is entered on the command line, the name of the member you were editing is assumed. To save the member under a different name, enter the new member name in the command line before pressing PF11. If the new member name already exists, an error message displays. You can save a portion of a member by making the first line you want to save the top line and entering the filename, followed by the number of lines you want to save. For example, if you want to save 20 lines of member TEST beginning with the twentieth line, you first position the member so that the first line you want to save is the top line on the screen; then enter TEST,20 on the command line and press PF11. You can use the partial save feature to save a portion of one member and then later use the GET command to copy it into another member.
PF12	Submits a member to POWER.
CLEAR key	Refreshes the screen and causes the editor to ignore anything entered on the current screen. Use the CLEAR key when you enter something you did not intend.

Compiling the REXX IMOD

When you have added or edited an IMOD and are ready to save it, you can either save it or file it.

You must file the member to compile it. Filing the member compiles it and saves the object code as NAME.OAt. The source is saved as NAME.OAL in the SYS$MON PDS.

Compiling translates REXX into executable code. Saving the member only stores the member in the SYS$MON PDS.

Saving an IMOD

To save an IMOD, use PF11 or the SAVE command.

If you use the PF11 key and you want to save the member to a name other than the current one, enter the new member name on the command line before pressing PF11. Otherwise, the member is saved to its old name.

When you are finished editing the IMOD and you want to save the member, enter SAVE on the command line and press Enter. The member name of the member you were editing is saved.

To save the member under a new name, enter the command on the command line and press Enter. If the member name specified already exists, an error message appears and you must use a different name.

```
SAVE new_member_name
```

Saving an IMOD does not compile the IMOD. Only a FILE command compiles an IMOD.

Filing an IMOD

To FILE an IMOD, use the FILE command.

When you are finished editing the IMOD and you want to file the member, enter FILE on the command line and press Enter. The member name of the member you were editing is saved and compiled.

To file the member under a name other than the current name, enter the command on the command line and press Enter.

```
FILE new_member_name
```

If the member name specified already exists, an error message appears and you must use a different name.

Determining the Member/User Name

The name of the currently selected member is displayed when you enter a question mark (?) on the command line. The name displayed is the one used if PF11 is pressed. This name is normally the name of the member entered at the start of the edit session. Change it by supplying a member name with the PF11 or SAVE command as described previously.

The user ID of the current user is also displayed. This ID is used for security checking by member name.

Performing Error Processing

Compilation errors are detected at FILE time. All the REXX code is validated for syntax, and if no errors occur the compiled member is saved. If an error occurs, the line in question is displayed as the current line and highlighted. An error message is displayed in the command error describing the error. If time is not available to correct the error, save the member. If the member was compiled previously, its object member remains intact.

1. Review the highlighted the line; it is where the error occurred.
2. Review the message describing the error displayed on the command line. If you need additional help for the message, press PF1.
3. Correct the error.
4. Enter the FILE command again. If there is another error, repeat the process from Step 1.

The following items describe some specific errors:

- If the error is in the prefix area, all commands entered in the prefix area for that screen are written back and suffixed with a question mark. No processing occurs until all commands are correct.
- When a multi-part command is entered and a part of the command is missing, the partial command is remembered and displayed until the entire command is entered. No command is processed in the prefix area until the partial command is completed. At that time, all the commands are processed.
- If a DELETE, COPY, or MOVE overlaps another command, it is considered either an error (if a number is was specified with the line command) or an incomplete command (if a multi-part command was entered).

Chapter 2
REXX Language

This chapter introduces the REXX language. REXX is a general-purpose structured language that has programming instructions and functions. REXX IMODs are compiled, translated, and executed.

Overview

A REXX program is built from a series of clauses that contain instructions, functions, and commands. Clauses can include terms such as function calls.

The BIM implementation of REXX includes the following:

- Instructions
- General REXX function calls
- BIM-FAQS/ASO-specific function calls
- BIM-FAQS/PCS-specific function calls
- User-written function calls

REXX Instructions

An *instruction* in REXX is one or more clauses, the first of which starts with a keyword that identifies the instruction. Some instructions affect the flow of control, while others provide services to the programmer. Some instructions, such as DO, include nested instructions.

REXX Functions

General REXX *functions* are also available. BIM REXX employs both general REXX functions and product-specific functions. Functions comprise a series of instructions that can receive data, process that data, and return a value to the IMOD that issued the function.

Functions differ from instructions because a function performs a process and returns the result of the process to REXX. An instruction is a single-step process.

User Functions

Any IMOD can be called as a function through a call or a function. You can also write internal functions within an IMOD. It is useful to set up commonly used routings or functions in their own IMODs, because it reduces complexity and saves coding time.

For example, the $CYCLE IMOD calls $CICSREP (provided at installation). $CICSREP issues the reply to CICS even if it is under ICCF. There is I/O overhead associated with calling another REXX IMOD since the new member will have to be read from disk. However, this is only noticeable in repetitive loops when an IMOD is referenced over and over again.

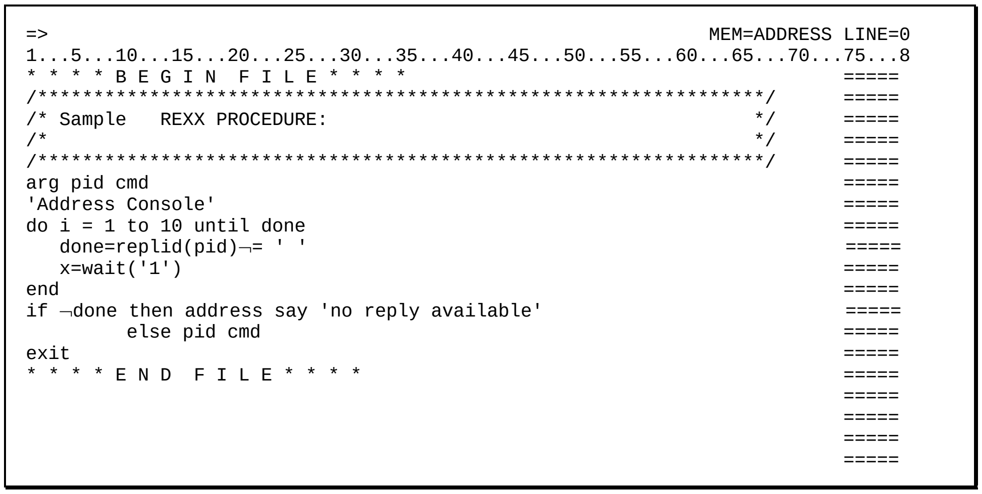
Using Online Help

The BIM editor has an extensive online help facility. You can access help on any REXX instruction or function directly from the member text. This helps the more experienced user who knows the command but may be unsure of the actual parameters. Also provided is a HELP menu panel that lists every REXX instruction, function, and operator. From this panel you can also obtain editor help by pressing PF1.

Direct Help

To obtain help on any REXX instruction or function that is coded in the IMOD, place your cursor on the first letter of the function you want explained and press PF1. A help panel appears that displays the help text of that function or instruction. If the BIM help processor cannot determine what function you desire help for, the Help Menu panel is displayed.

The following screen is an example of a REXX IMOD that contains the instruction *DO*.

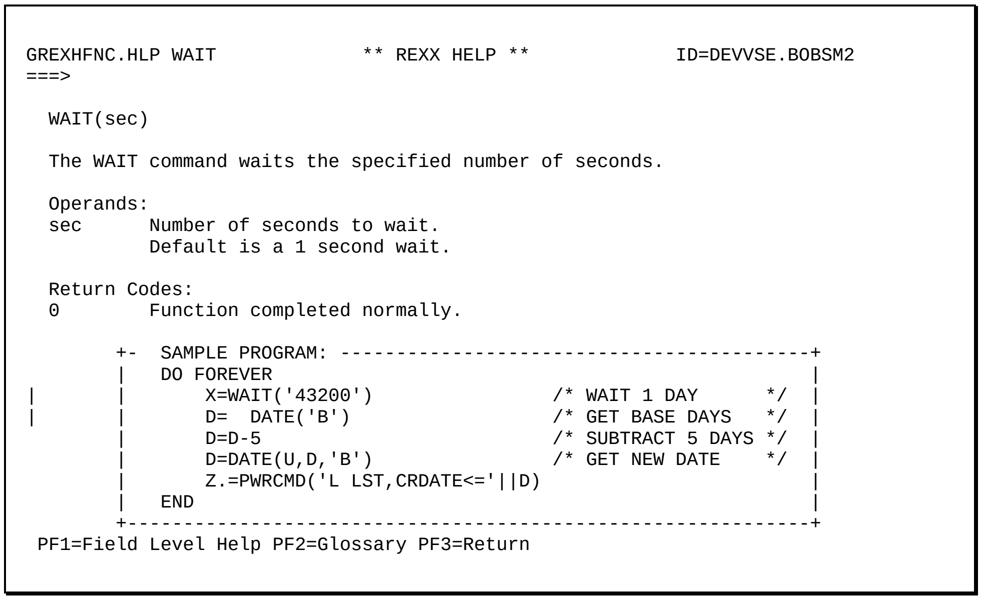


```
=>                                                              MEM=ADDRESS LINE=0
1...5...10...15...20...25...30...35...40...45...50...55...60...65...70...75...8
* * * * B E G I N   F I L E * * * *                                        =====
/*****************************************************************/      =====
/* Sample   REXX PROCEDURE:                                     */      =====
/*                                                              */      =====
/*****************************************************************/      =====
arg pid cmd                                                                =====
'Address Console'                                                          =====
do i = 1 to 10 until done                                                  =====
   done=replid(pid)¬= ' '                                                  =====
   x=wait('1')                                                             =====
end                                                                        =====
if ¬done then address say 'no reply available'                             =====
         else pid cmd                                                      =====
exit                                                                       =====
* * * * E N D   F I L E * * * *                                            =====
                                                                           =====
                                                                           =====
                                                                           =====
                                                                           =====
```

Sample REXX IMOD Definition Screen

Accessing Online Help from REXX IMOD Editor

To access help for the WAIT( ) function, move your cursor to the WAIT function. Then press PF1 to display the following BIM REXX help screen.



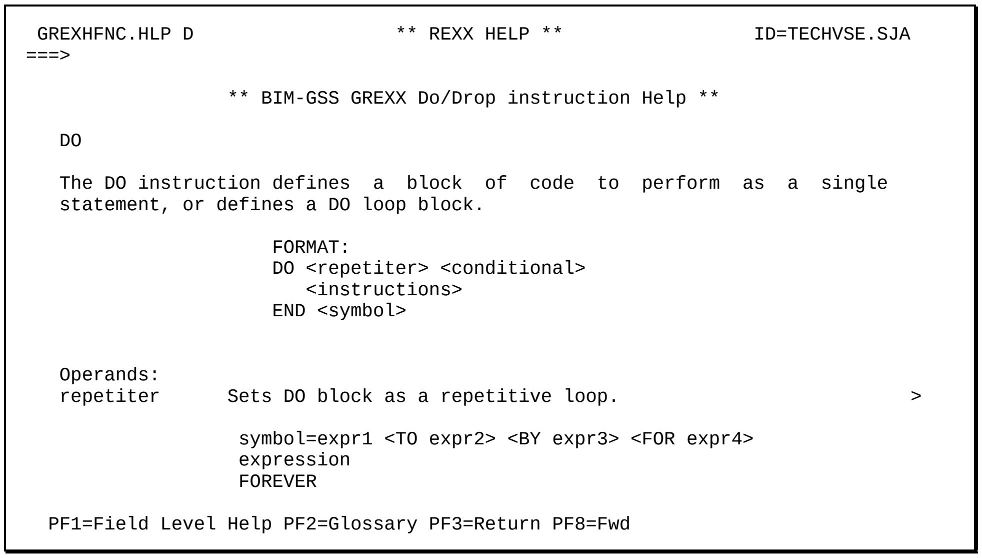
```
GREXHFNC.HLP WAIT             ** REXX HELP **             ID=DEVVSE.BOBSM2
===>

  WAIT(sec)

  The WAIT command waits the specified number of seconds.

  Operands:
  sec      Number of seconds to wait.
           Default is a 1 second wait.

  Return Codes:
  0        Function completed normally.

        +-  SAMPLE PROGRAM: -----------------------------------------+
        |   DO FOREVER                                               |
|       |       X=WAIT('43200')                /* WAIT 1 DAY      */  |
|       |       D=  DATE('B')                  /* GET BASE DAYS   */  |
        |       D=D-5                          /* SUBTRACT 5 DAYS */  |
        |       D=DATE(U,D,'B')                /* GET NEW DATE    */  |
        |       Z.=PWRCMD('L LST,CRDATE<='||D)                        |
        |   END                                                      |
        +------------------------------------------------------------+
 PF1=Field Level Help PF2=Glossary PF3=Return
```

Sample REXX Help Screen

Accessing Additional Help from Help Screen

The following screen is an example of the DO help that displays if you place your cursor on *DO* in the help example and press PF1. By pressing PF2, you can move to the glossary of terms and definitions.

In this screen, the words *DO* and *DATE* are highlighted or red. You can tab to these fields and press PF1 for more help. Additionally, you can nest level after level of help and press PF3 to return through the nest, or press PF12 to return to the original base help panel.

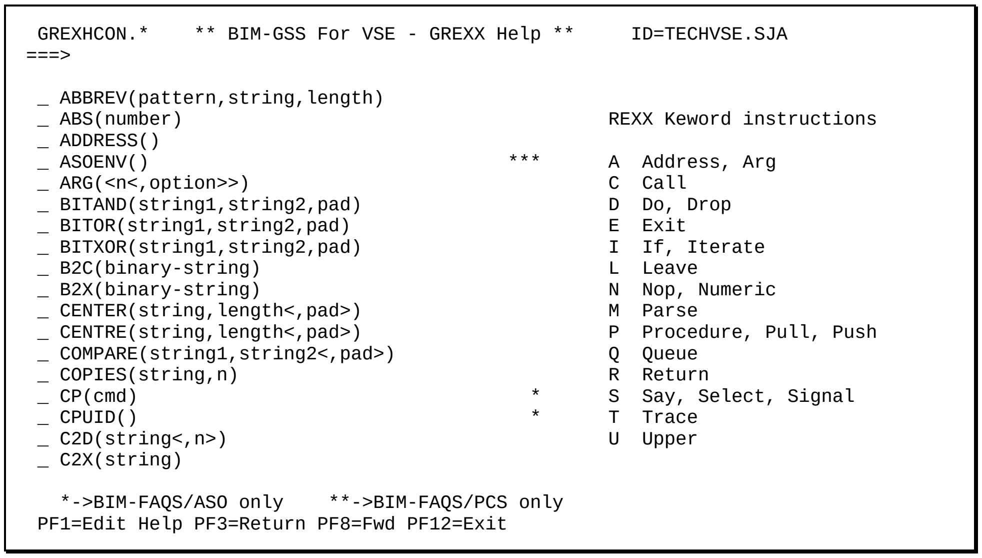


```
 GREXHFNC.HLP D                  ** REXX HELP **                 ID=TECHVSE.SJA
===>

                  ** BIM-GSS GREXX Do/Drop instruction Help **

   DO

   The DO instruction defines  a  block  of  code  to  perform  as  a  single
   statement, or defines a DO loop block.

                      FORMAT:
                      DO <repetiter> <conditional>
                         <instructions>
                      END <symbol>


   Operands:
   repetiter      Sets DO block as a repetitive loop.                          >

                   symbol=expr1 <TO expr2> <BY expr3> <FOR expr4>
                   expression
                   FOREVER

  PF1=Field Level Help PF2=Glossary PF3=Return PF8=Fwd
```

Sample DO Help Screen

BIM REXX Help Menu

To access the BIM REXX Help Menu while you are editing an IMOD, press PF1 with the cursor on the command line. The following screen shows a sample BIM REXX Help Menu.

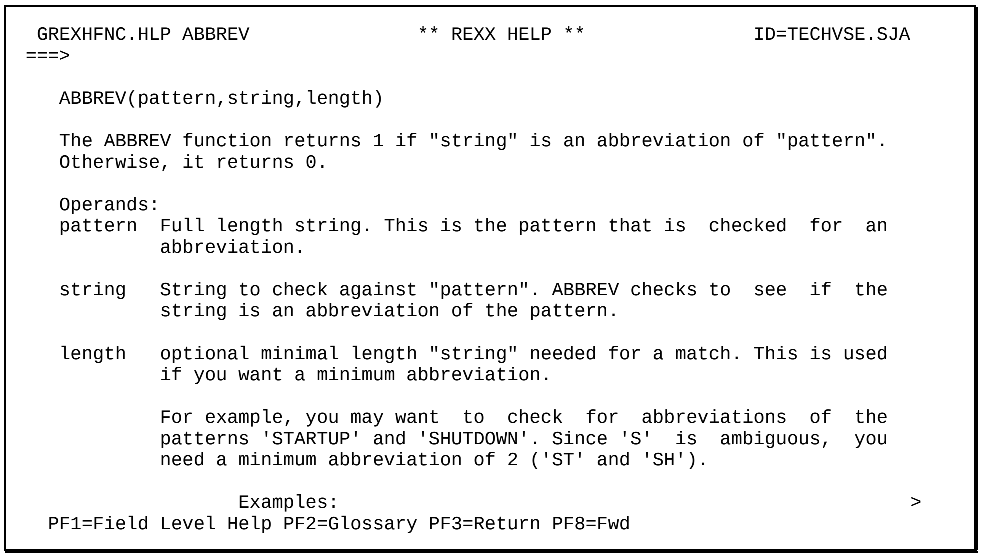


```
 GREXHCON.*    ** BIM-GSS For VSE - GREXX Help **     ID=TECHVSE.SJA
===>

 _ ABBREV(pattern,string,length)
 _ ABS(number)                                       REXX Keword instructions
 _ ADDRESS()
 _ ASOENV()                                 ***      A  Address, Arg
 _ ARG(<n<,option>>)                                 C  Call
 _ BITAND(string1,string2,pad)                       D  Do, Drop
 _ BITOR(string1,string2,pad)                        E  Exit
 _ BITXOR(string1,string2,pad)                       I  If, Iterate
 _ B2C(binary-string)                                L  Leave
 _ B2X(binary-string)                                N  Nop, Numeric
 _ CENTER(string,length<,pad>)                       M  Parse
 _ CENTRE(string,length<,pad>)                       P  Procedure, Pull, Push
 _ COMPARE(string1,string2<,pad>)                    Q  Queue
 _ COPIES(string,n)                                  R  Return
 _ CP(cmd)                                    *      S  Say, Select, Signal
 _ CPUID()                                    *      T  Trace
 _ C2D(string<,n>)                                   U  Upper
 _ C2X(string)

   *->BIM-FAQS/ASO only    **->BIM-FAQS/PCS only
 PF1=Edit Help PF3=Return PF8=Fwd PF12=Exit
```

Sample REXX Help Menu

Function Help

To access information for a particular REXX function or keyword, select that item and press Enter to display the Help information. The following screen shows an example of the help information for the ABBREV function.



```
 GREXHFNC.HLP ABBREV               ** REXX HELP **               ID=TECHVSE.SJA
===>

   ABBREV(pattern,string,length)

   The ABBREV function returns 1 if "string" is an abbreviation of "pattern".
   Otherwise, it returns 0.

   Operands:
   pattern  Full length string. This is the pattern that is  checked  for  an
            abbreviation.

   string   String to check against "pattern". ABBREV checks to  see  if  the
            string is an abbreviation of the pattern.

   length   optional minimal length "string" needed for a match. This is used
            if you want a minimum abbreviation.

            For example, you may want  to  check  for  abbreviations  of  the
            patterns 'STARTUP' and 'SHUTDOWN'. Since 'S'  is  ambiguous,  you
            need a minimum abbreviation of 2 ('ST' and 'SH').

                   Examples:                                                  >
  PF1=Field Level Help PF2=Glossary PF3=Return PF8=Fwd
```

Sample ABBREV Function Help Menu

REXX General Usage

A simple yet powerful language, REXX is easy to use and to learn. Most functions are very natural and similar in syntax. To aid in implementing BIM REXX, BIM provides many practical examples in the installation and an extensive online help facility.

Comments

REXX allows comments, and all REXX programs must start with a comment. A comment is delimited by a `/*` and an `*/`. Since BIM REXX implementation is compiled, heavy use of comments is encouraged. Comments do not add overhead in a compiled version of REXX.

Comments are free-form, appearing anywhere and spanning any number of lines. Comments are useful for documenting IMODs and for commenting out sections of code.

```
/* This is a comment required as the first line of an IMOD */
arg data
say data
/*
do i =1 to 10            /* loop for 10            */
   z.i=word(data,i)      /* set stems for each word*/
   if word='' then leave /* if no more word leave  */
end
*/
exit I
```

In the previous example, the DO loop is commented out and is not executed. Notice how a comment can enclose another comment.

Symbols

A symbol is a variable, constant, or keyword made up of the following characters: A-Z, a-z, 0-9, period (.), exclamation mark (!), underscore (_), at sign (@), pound sign (#), question mark (?), and dollar sign ($). Symbols are translated to uppercase before use. For example, the symbols BOB, Bob, and bob are identical.

Strings

A string is a group of characters delimited by single quotes (' ') or double quotes (" "). You can delimit the string with double quotes if you include a single quote in the string, or vice versa. When this is not possible, place two single or two double quotes together to denote a single character. The null string is used many times in REXX and is shown as ' '.

```
'This is a test'
"Don't go in the Basement"
'Don''t go in the "Basement"'
''                                  /* null string */
```

Binary Strings

A binary string is a string of 0's and 1's, grouped in four characters that can be delimited by one or more blanks. The first string of characters is assumed to have a length of 4, and it is right justified with zeros added to align the string to four characters. The string must be delimited on the left with a single quote (') or double quote ("), and on the right with a 'B, 'b, "B, or "b.

```
'001 0000'b     ==>  '10'x
'0010000'b      ==>  '10'x
'11110000'b     ==>  'F0'x
'11 0000 1111'b ==>  '30F'x
'11 00001111'b  ==>  '30F'x
'00110001'b     ==>  '31'x
```

Hexadecimal Strings

A hexadecimal string is a string made up of the characters 0-9, A-F, a-f, grouped in pairs delimited by one or more blanks. The first character does not have to be paired, and a 0 is added on the left to pair this character. The string must be delimited on the left with a single quote (') or double quote ("), and on the right with an 'X, 'x, "X, or "x.

```
'F c7 F8'x      ==>'0FC7F8'x
'C1'x           ==>  'A'
'C1c2C3'x       ==>  'ABC'
'123 45'x       ==>  '012345'x
```

Expressions

An expression is an instance of one or more strings, symbols, operators, or functions. Expressions are evaluated left to right with respect to parentheses and operator precedence.

```
'aaa'||'bbb' time()
x * y
word(substr(x),5)
```

Assignments

Variables can be assigned data by the use of ARG, PARSE, PULL, and ADDRESS environments. You can also use an equal sign (=) to assign a symbol to the value of an expression.

```
symbol=expression
```

REXX Operators

Prefix Operators

+	Positive number
-	Negative number

Boolean Operators

&	And
|	Or
&&	Exclusive Or
¬	Not

Algebraic Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Remainder of division
%	Integer division
||	Concatenation

Comparators

=	Equal
==	Identical
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
¬=	Not equal
<>	Not equal
><	Not equal
¬==	Not identical
¬>	Not greater than
¬>=	Not greater than or equal
¬<	Not less than
¬<=	Not less than or equal

Variables

BIM implementation of REXX contains four types of variables:

- Simple variables
- Stem variables
- Global variables
- Global stem variables

Simple Variables

Simple variables are symbols whose values can be changed during the execution of a REXX IMOD. Simple variables can be up to 50 characters long and can be assigned values up to 4096 bytes. These variables are local to the currently executing REXX IMOD; they cannot be referenced by other IMODs or procedures. However, called procedures can access these simple variables through the EXPOSE command on the procedure definition.

Stem Variables

Stem variables are composed of a stem symbol and a period (.), denoting a family of stems that can be cleared, initialized, set, or dropped.

```
A.=''           /* set the stem family A. to null      */
data.='none'    /* set the stem family data. to 'none' */
data.=cp('ind') /* set the stem family data. to cp data*/

drop b.         /* drop the stem family b.             */
```

You can assign or reference any member of a stem family by appending a symbol to the family name. This symbol can be a constant or a variable. A variable is useful for accessing all members of a family when the members are named by numbers and the 0 member contains the number of members in the family. Some functions assign stems in this manner.

```
A.1='test'
test.i='the' i||'th member'

family.member='john'
```

Stem Variable Assignments

Some functions return stem variable assignments. Stem variable assignments indicate that a variable's value changes during the execution of REXX and that a new value is assigned to it. The functions CP, POWER, PWRCMD, and MESSAGE return stem variable assignments.

```
z.=cp('IND')
 do i = 1 to z.0
    say z.i

 end
```

Global Variables

Global variables are symbols whose values can be changed during the execution of a REXX IMOD. Global variables can be up to 8 characters long and can be assigned values up to 105 bytes, including the variable name. Global variables are prefixed with an ampersand (&). Because of the nature of REXX, global variables cannot be concatenated through abuttal; they must be concatenated with a blank or ||.

Global variables can be used only for REXX IMODs executed through the FAQSAO task. These IMODs include those triggered using SMSGs, console messages, console commands, online commands, browser requests, BIM-FAQS/PCS commands, and GEM (Global Event Manager). Global variables are kept until you assign them to null or drop them. They are kept on disk and in storage and are unaffected by an IPL.

Because of the nature of the FAQSAO task multi-threading IMODs, you can be assured of the state of a global variable until you perform a function that has an implied wait—such as CP(), WAIT(), POWER(), PWRCMD, MESSAGE(), or MSG(). At these implied waits, other IMODS are run, and the values of your global variables are subject to change.

```
&A=''

&node='enabled'
```

Global Stem Variables

Global stem variables function like normal stem variables but according to the rules of global variables. The only difference between global variables and global stem variables is their life expectancy. Since global stem variables are not written to disk, they do not survive end-of-job.

How Arguments Are Passed

The method for passing arguments created by commands, messages, or SMSGs to BIM-FAQS/ASO REXX IMODs is SAA-compatible. The information created by the command, message, or SMSG is passed to the IMOD. All related data can be accessed using the ASOENV function. ASOENV is designed to return information that previously passed as part of the argument.

Default Argument Passing Method

The IMODs, commands, and action files shipped with your installation tape use the new argument settings. Any calls to IMODs prefixed with $ and downloaded at installation must be modified to use the new argument settings. Also, the ASO CMS EXEC used to call IMODS through SMSG has changed to support the new ASO identifier, which denotes the new argument settings.

You still have the option of using the old argument settings. Fields are provided on the action and command file definition screens that control the argument setting. By default, these fields are enabled for actions and commands already defined.

If a Message Triggered an IMOD

Prior to BIM-FAQS/ASO version 3.x, when a message triggered an IMOD the following information was passed to the IMOD:

- Action name
- Partition ID
- Jobname
- Phase name
- Time
- Message that triggered the IMOD

Now only the message that triggered the IMOD is passed to the IMOD.

If a Command Triggered an IMOD

Prior to BIM-FAQS/ASO version 3.*x*, when commands triggered an IMOD the following information was passed to the IMOD:

- Command name
- Related data that triggered the execution of the IMOD

Now only the data that triggered the IMOD is passed.

If an SMSG Triggered an IMOD

Prior to BIM-FAQS/ASO version 3.x, when an ASO CMS EXEC triggered IMODs using SMSG, the CMS machine ID and related data were passed to the IMOD. Now only related data is passed when an SMSG triggers an IMOD.

Differences Between BIM REXX and IBM REXX

The following section lists all differences between the B I Moyle Associates Inc. and IBM's implementations of REXX. The REXX used as the basis for comparison is TSO/E REXX, plus certain clarifications from M. F. Cowlishaw's *The REXX Language: A Practical Approach to Programming*. BIM REXX includes a number of very subtle and minor differences. The only significant lack in BIM REXX implementation is the lack of the INTERPRET keyword (because BIM uses a compiled REXX implementation).

INTERPRET

INTERPRET is not supported. However, the VALUE function can usually perform the same function as INTERPRET.

Loop Control Variables

Loop control variables can be altered in BIM REXX, but the name of the variable cannot be altered. IBM REXX permits constructions like the following:

```
DO a.i=1 to 4;i=x;end
```

a.i is the loop control variable, whose name is altered by changing the variable *i* within the loop. IBM REXX supports this by searching the symbol table each cycle, with very large overhead. BIM REXX saves a pointer to the variable information at the start of the loop, so it ignores any change of variable name such as the above. Both IBM REXX and BIM REXX permit the value of the loop control variable to be altered or referenced within the loop. This practice is strongly discouraged, but both IBM REXX and BIM REXX allow it.

Labels

Labels in BIM REXX cannot be duplicated. IBM REXX permits duplicate labels and ignores all but the first. BIM REXX signals duplicate labels with a compile time error message.

Labels in BIM REXX are restricted in length. IBM REXX supports function names and labels of the same length as symbols, and supports all symbols up to 250 characters in length. BIM REXX restricts the length of a function label or a called label to 32 unique characters. Additional characters in function labels can be supplied but are ignored. All other BIM REXX labels can contain 250 characters.

Signal

IBM REXX does not support SIGNAL into a DO block. BIM REXX disallows SIGNAL into an active DO group, but it does not error flag signals into a simple DO block. An active DO group is considered to be any active looping DO group defined by DO UNTIL, WHILE, or TO/BY/FOR. A signal into a simple DO block with no iteration is not an error.

MAX and MIN

MAX and MIN support only whole numbers. IBM REXX allows these functions to operate on decimal numbers.

Symbols

Symbols can include additional characters. BIM REXX permits symbols to contain alphanumerics, underscore (_), at sign (@), pound sign (#), question mark (?), exclamation mark (!), and dollar sign ($). These are the same characters allowed in symbols by TSO/E REXX. IBM REXX does not allow @, #, and $ to be used.

TRACEs

TRACEs are not implemented in a totally compatible fashion. REXX supports an extensive TRACE facility. BIM REXX supports a slightly different and extended TRACE facility.

BIM REXX supports the TRACE All, Commands, Error, Failure, Intermediate, Labels, Normal, Off, and Results operands of SAA REXX. The trace output from BIM REXX can differ slightly from REXX trace output.

For example, BIM REXX does not display all continued lines, although it does display the first line of a group of continued lines. BIM REXX supports the TRACE VALUE form for the above commands. BIM REXX also supports TRACE VAR vlist, where "vlist" is a list of names of variables for which alterations are traced. BIM REXX also supports TRACE VAR (vname), where "vname" is an indirect reference to a list of variables—like the "DROP (vname)" support. If a stem root is specified as the operand of this form of the trace, then it and all subsequently created stem items are traced. This variable alteration trace is not currently supported by IBM REXX, but it has been a SHARE requirement for SAA REXX. If IBM supports it in the future, BIM REXX will comply with the IBM implementation. ***This means that the BIM REXX implementation is subject to change.***

C2D

In BIM REXX, the C2D function is restricted to four input characters. IBM REXX permits you to pass a string of any length to the C2D function. BIM REXX passes a string of any length to C2D when the second operand to the function is omitted, but it limits the string to four characters when the second operand is specified.

Floating Point Numbers

BIM REXX implements most standard REXX floating point arithmetic. BIM does not implement the fourth and fifth operands of the FORMAT function (expp and expt). NUMERIC FORM SCIENTIFIC is assumed. No NUMERIC FORM statement is supported.

UPPER

UPPER is supported for stem root variables in BIM REXX. In IBM REXX, UPPER A. is an error. In BIM REXX, it is supported by converting the value of A. to uppercase.

DATE

You can code DATE(Option,yyyyddd) to convert the specified date to the format specified by Option. You can code DATE(Option,ddddddd,'B') to convert the specified base day number to the format specified by Option. You can code DATE(Option,yyyymmdd,'S') to convert the specified standard date number to the format specified by Option.

DUMPSTG

DUMPSTG(address,length) is similar to the storage function except that the designated data is returned as display hexadecimal. The maximum length supported is 256 bytes (512 bytes of display hex).

Performance Hints

The following list summarizes performance techniques for BIM REXX and includes items unique to BIM REXX as well as some items common to BIM and IBM REXX.

Global Variables

Global variables use more storage and are slightly slower to access and update than local variables.

Stem Variables

Stem variables use more storage and are slower to access and update than non-stem variables.

Assignments = vs ==

The normal relational equal operator (=) is substantially slower than the relational identical operator (==); therefore, you should use "==". This is an extremely important point for both IBM REXX and BIM REXX.

Long Strings

Long strings can be much slower than strings < 500 characters. Concatenation to build strings longer than the default temporary string size is not optimized and can be slow.

SUBSTR

SUBSTR with explicit numeric 2nd and 3rd operands is optimized at compile time and is very fast.

Numbers

Numbers expressed as simple integers instead of as strings (with quotes) are handled as binary values. They are much faster to use in arithmetic than strings. This internal use of binary numbers by BIM REXX is transparent to the user and only affects performance. BIM REXX maintains a value as internal binary once created until it is concatenated or displayed when the value is converted to character. This conversion process is relatively slow.

Numbers with more than 9 digits or numbers expressed as floating point values are much slower than integers that fit in a full word. The default NUMERIC DIGITS value is 9. Any NUMERIC DIGITS value other than 9 seriously impacts performance.

Comments

Comments cost nothing and can be used freely. In IBM REXX, comments can be expensive.

Statements

Statements can occupy the same or separate lines with no cost. In IBM REXX, statements coded on one line are faster.

Arithmetic Items

Arithmetic items are best limited to integers that can be expressed as a binary full word. BIM REXX supports floating point numbers up to 250 digits for all operations. BIM REXX also uses binary arithmetic and carries numeric items in binary for all operations where this is possible. BIM REXX optimizes all one- and two-digit numerics, which are extremely quick and require no temporary storage (other numbers can use temporary storage).

Chapter 3
REXX Instructions

This chapter lists and explains REXX instructions.

An *instruction* in REXX is one or more clauses, the first of which starts with a keyword that identifies the instruction. Some instructions affect the flow of control, while others provide services to the programmer. Some instructions, such as DO, include nested instructions. REXX instructions do not return data and cannot be used in an assignment.

ADDRESS environment <command>

Purpose

The ADDRESS instruction is used to pass a command to a specific processor. *environment* must be specified. *command* is optional and if not specified, the ADDRESS *environment* is set. Once an environment is set, any expressions not understood by REXX are passed to the current environment. If *command* is specified, that command is passed to the temporary environment specified.

Example

```
                              Required product
ADDRESS AO                        ASO PCS
ADDRESS CARD
ADDRESS CICS                          PCS
ADDRESS CONSOLE
ADDRESS DISK
ADDRESS EPIC                         EPIC
ADDRESS EVENT                         PCS
ADDRESS EVSE                   CA-EXPLORE
ADDRESS OUTPUT
ADDRESS PCL                           PCS
ADDRESS PCS                           PCS
ADDRESS PDS                        ASO PCS
ADDRESS PDATE                         PCS
ADDRESS POWER                     ASO PCS
ADDRESS PROGRAM                   ASO PCS
ADDRESS PUNCH
ADDRESS SYS                       ASO PCS
```

ARG template

Purpose

The ARG instruction retrieves arguments passed to a function or procedure. ARG is a shorthand for the PARSE instruction.

Example

```
ARG arg1 arg2
```

RESULT

Purpose

The RESULT variable is set whenever a function is called and returns a value (an online function call is replaced by the function's result, but a called function needs a place to put its value). RESULT is also set when ADDRESS returns a value. The RESULT variable stays set until another function CALL or an ADDRESS command changes its value. You can refer to RESULT anywhere a variable can be referenced.

Example

```
result=''
call set
say 'set returned' result
exit result

set:
x = 'data'
return result

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
set returned data
```

CALL function <expression<,expression>...>

Purpose

CALL invokes a function or procedure that exits using a RETURN statement.

Example

```
CALL fact(4)
    ...

FACT: PROCEDURE
       PARSE ARG n
       if n=1 then return n

       RETURN n*(FACT(n-1))
```

The symbol RESULT is used to store any returned value of a called routine. In the above example, RESULT would receive the value '24'. The above call could have been written as the following with equivalent results.

```
result=fact(4)
    ...

 FACT: PROCEDURE
        PARSE ARG n
        if n=1 then return n
        RETURN n*(FACT(n-1))
```

DO

Purpose

The DO instruction defines either of the following:

- A block of code to perform as a single statement
- A DO loop block

Syntax

```
DO <repetiter> <conditional>
   <instructions>

END <symbol>
```

Operands

repetiter	Sets DO block as a repetitive loop. **symbol=expr1 <TO expr2> <BY expr3> <FOR expr4>** **expression** **FOREVER**
conditional	Defines exception conditions to break or continue loops. Expression is any expression that evaluates to 1 or 0. See also sections LEAVE and ITERATE. **WHILE expression** **UNTIL expression**
instructions	Any instructions you want to be executed in the DO group.

Examples: Simple Do Groups

```
If 'A'='A' then do
    say 'A=A'
    a='A=A'
end
else do
     say 'A=B'
     a=b
end
```

Examples: Simple Do Loops

```
Do 5
   i=i+1
   say i
end
do i = 1 to 5
    say i
end
do forever
    i=i+1
    say i
    x=wait('60')
end
```

Examples: Complex Do Loops

```
do i = i to i+5
    say i
end
Do i= 10 to 1 by -1
   say i
end
do i = 1 to 5  until i=4
    say i
end
do forever while i<=1000
    i=i+1
    say i
    x=wait('60')
end
done=0
do forever until done
   z=wait('1')
   done=replid(pid)¬=' '
end
```

DROP

Purpose

The DROP instruction discards a variable or variables when you no longer need them. When a stem root is dropped, all stem values are discarded. In the following example, *(vname)* indirectly references variables to drop. The main use of DROP is to save storage or destroy a global variable. Setting a variable to null is not equivalent to dropping the variable.

Example

```
DROP a. b c (vname) &jobname.bg
```

EXIT <expression>

Purpose

The EXIT instruction ends execution of a REXX program. The optional operand can be used to have a result returned. You should use RETURN for end-user and internal functions. EXIT causes immediate termination of the whole program. An EXIT is assumed at the end of any IMOD.

Example

```
I=10

exit 'value of i='i
```

'value of i=10' is returned as the program result.

IF

Purpose

The IF instruction performs a relational test and conditionally executes REXX statements based on the result of the test. The statement executed can be a single instruction or a DO group.

Example

```
IF i=1 THEN say 'i was 1'
       ELSE call fact(i)
IF i=1 & i>0 THEN do
                  say i
                  i=i+1
               end
       ELSE  do i = i to i+5
                 say i
               end
done=0
if ¬done then x=wait(4)
```

ITERATE <symbol>

Purpose

The ITERATE instruction continues with a DO loop from within that loop. If no symbol is specified, the innermost loop is iterated. If a symbol is specified, it must match the name of the control variable symbol of a currently active DO loop.

Examples

```
do i=1 to 4
   if i=3 then iterate
      say 'i='||i
end
```

Output is i=1, i=2, and I=4. The i=3 line is not output.

```
do j=1 to 5
   say 'j='||j
   do i =1 to 5
      if i*j=6 then iterate
      say 'i='||i i*j
      if i*j=8 then iterate j
   end
end
```

THEN

Purpose

THEN is a normal success path for IF instruction.

Example

```
If i=1 then say 'i was 1'
```

ELSE

Purpose

ELSE is an alternate path for IF instruction. The ELSE clause is executed if the condition tested by the IF instruction fails.

Example

```
i=2
if i=3 then say 'If test succeeded'
       else say 'i='||i
```

Output is i=2. The 'If test succeeded' line is not output.

LEAVE <symbol>

Purpose

The LEAVE instruction exits a DO loop or DO block without completing it. If no symbol is specified, the innermost loop is left. If a symbol is specified, it must match the name of control variable symbol of a currently active do loop.

Example

```
do i=1 to 4
   if i=3 then leave
   say 'i='||i

end
```

Output is i=1, i=2. The i=3 and i=4 lines are not output. The LEAVE command causes an immediate exit from the loop.

```
do j=1 to 5
   say 'j='||j
   do i =1 to 5
      if i*j=6 then leave
      say 'i='||i i*j
      if i*j=8 then leave j
   end
end
```

NOP

Purpose

The NOP instruction is a dummy instruction. NOP has no effect. It may make a listing more readable. It is also necessary for some complex IF-THEN-ELSE structures.

Example

```
do i=1 to 4
if i=3 then nop
       else say 'i='||i

end
```

Output is i=1, i=2, and i=4. The i=3 line is not output.

NUMERIC

Purpose

The NUMERIC instruction allows you to set the floating point precision of arithmetic operations and comparisons. The current setting can be checked using the FUZZ function.

Example

```
NUMERIC FUZZ number
```

OTHERWISE

Purpose

The OTHERWISE clause is an optional clause of the SELECT instruction. If OTHERWISE is omitted and a WHEN does not succeed, an execution error is signaled. It is strongly suggested that you always code an OTHERWISE. You can code NOP as the target of the OTHERWISE.

Example

```
select
  when a=1 then i=1
  when a=2 then i=0
  otherwise say 'a is neither 1 nor 2'
end
```

PARSE

Purpose

The PARSE instruction scans a string and extracts fields directed by a pattern.

Examples

```
PARSE <UPPER> ARG                         <template>
              PULL
              SOURCE
              VALUE      expression WITH
              VAR        name
              VERSION
```

```
PARSE <UPPER> ARG <template>
```

Parses the string passed to the IMOD, subroutine, or procedure with the provided template.

```
parse arg w1 . w3 rest
```

The 1st argument is scanned and w1 receives the 1st word; word 2 is ignored; w3 receives the 3rd word; and the remainder is placed in rest.

```
PARSE <UPPER> source template
```

Not supported.

```
PARSE <UPPER> PULL template
```

The top stack item is uppercased and put in *value*. The stack must not be empty. Use the QUEUED( ) function to check that the stack is not empty.

```
PARSE <UPPER> VAR symbol template
parse var vname a1 '.' a2
```

The value of *vname* is scanned for a period and the data up to the period is placed in a1, the data following in a2. The period is deleted, and it is not placed in either variable. Blanks and binary zeros are not stripped.

PROCEDURE <expose <(> variable list <)> >

Purpose

The PROCEDURE instruction defines a REXX subroutine with its own set of variables. PROCEDURE hides all preexisting variables.

You can add an EXPOSE clause to reveal selected variables.

Examples

```
a=1;b=2
call sub a b
/* a still equal 1 */
say result
exit

sub: procedure
arg a b
a= a+b
return a
```

```
a=1;b=2
call sub  /* returns 4 */
say 'sub returned' result
exit

sub: procedure expose a,b
if a=1 then return b+2
       else return 0
```

PULL

Purpose

The PULL instruction reads and parses the next string from the program stack. *PULL name* places the top stack item in *name*. *PULL template* is short for PARSE UPPER PULL *template*.

PUSH

Purpose

The PUSH instruction places its string operand at the top of the program stack. PUSH maintains a LIFO (Last In First Out) stack. Any expression can be used as the operand of PUSH.

Example

```
do i=1 to 4
   push 'i='i
end
```

QUEUE

Purpose

The QUEUE instruction places a string at the bottom of the program stack. QUEUE maintains a FIFO (First In First Out) stack. Any expression can be used as the operand of QUEUE. Use the PULL instruction to retrieve the stack contents.

Example

```
do i=1 to 4
   queue 'i='i
end
```

RETURN

Purpose

The RETURN instruction returns control from a function or called routine. Any expression can be used as the operand of RETURN. For a function, an expression is required as the operand, and it is the value of the function. For a call, an expression is optional as the operand, and it is returned as the value of the RESULT variable.

SAY

Purpose

The SAY instruction displays its operand to the user. The target varies depending on the environment the IMOD is executed in.

Environment

Batch	Goes to SYSLOG unless ADDRESS AO SAY redirects it.
JCL	Goes to SYSLOG.
CMS	Goes to the terminal.
Console CMD	Goes to SYSLOG.
SMSG to VSE	Returns to the VM machine that issued the SMSG, unless address AO SAY is used to redirect to SYSLOG.
ONLINE	Returns to the online session that triggered the IMOD, unless address AO SAY is used to redirect to SYSLOG. Only 80 characters can be displayed with SAY. To display more, use the SUBSTR function.
HTTP (Browser)	**Goes to a defined buffer unless ADDRESS AO SAY redirects it. If redirected, use an ADDRESS AO SAY HTML to continue sending SAY data to the buffer.** By using ADDRESS AO SAY HTML you can generate an HTML document using SAY commands.
CICS/BATCH (BUFR)	**Goes to a defined buffer unless an ADDRESS AO SAY redirects it. If redirected, use an ADDRESS AO SAY BUFR to continue sending SAY data to the buffer.** **Only 80 characters may be displayed with SAY. To display more information, use the SUBSTR function.**

Example

```
say 'y='||y x*5 '.......' time()
say substr(longvar,1,80)
say substr(longvar,80,80)
```

SELECT

Purpose

The SELECT instruction defines a SELECT block with one or more WHEN clauses, and an optional OTHERWISE clause. SELECT blocks are more efficient than multiple IF THEN clauses or groups.

Example

```
select
    when x=y then ...
    when x<y  y<z then ...
    when x='test' then do
                    ...
                  end
    otherwise do
               ...
              end
end
```

SIGNAL

Purpose

The SIGNAL instruction causes a branch (abnormal change in logic flow). The required operand must be a label in the current program.

The first SAY instruction is not executed. The only output seen by the user is 'this is only output'.

Example

```
signal skipit
say 'you will never see this'
skipit:
say 'this is only output'
```

TRACE All/Commands/Error/Fail/Intermediate/Labels/Normal/Off/Results

Purpose

The TRACE instruction enables diagnostic displays to help with debugging a program. All SAA traces are supported plus TRACE VAR vname.

Example

```
trace var b
do i=1 to 4
   b=i*2

end
```

Output is 3:B=2, 3:B=4, 3:B=6, 3:B=8   (3 indicates the line number).

UPPER

Purpose

The UPPER instruction translates the value of each operand to uppercase. Any expression can be used as the operand of UPPER.

Example

```
a='abcd';b='UPPER ALREADY'
upper a b

say 'a='a',b='b
```

a=ABCD,b=UPPER ALREADY is the output.

WHEN

Purpose

The WHEN clause is a required clause of SELECT instruction.

Example

```
select
  when a=1 then i=1
  when a=2 then i=0
  otherwise nop
end
```

Chapter 4
REXX ADDRESS Environments

This chapter covers REXX ADDRESS environments, the environments in which REXX commands are processed.

ADDRESS AO

The ADDRESS AO command provides an interface to the FAQSAO IMOD processor. It allows you to control the way an IMOD functions on your system.

Syntax

```
ADDRESS AO  LOOP    USER limit
            LOOP    SYS  limit
            SAY
            SAY     CONSOLE
            SAY     BUFR
            SAY     BVER
            SAY     HTML
            SAY     HVER
            MSG
            MSGNOH
            DUMP
            PURGE
```

Environment

This ADDRESS environment is only available for REXX IMODs run through the FAQSAO task. This includes IMODs triggered through Console commands, Console messages, Online commands, SMSG, EVENTS, the HTML interface and started in CICS or a batch partition.

Operands

LOOP	The SYS option sets the default for all IMODs run through FAQSAO. The SYS option is normally set in the AO initialization panel. The User option is used when you have a long-running or loop-intensive IMOD. You can specify a limit of 1-999,999 or of * to run forever.
SAY	This option redirects any traces or SAYs to the requested location, which may be CONSOLE, an HTML document, or a BUFFER . This changes the default, which is to echo the results of the IMOD to the originator If you do not specify an

	option on the ADDRESS AO SAY request, SAY resumes default operation. The BVER and HVER are special requests and act as CLOSE routines for the ADDRESS AO SAY BUFR and ADDRESS AO SAY HTML environments respectively.
MSG	**VM only.** Set SAY to issue MSG user when triggered through SMSG. MSG is normally set in the AO initialization panel.
MSGNOH	**VM only.** Set SAY to issue MSGNOH user when triggered through SMSG. MSGNOH is normally set in the AO initialization panel.
DUMP	Issuing this command sets the extended dump flag on. DUMP is normally set in the AO initialization panel.
PURGE	Purge any IMODs that are waiting to begin execution. PURGE is normally reserved for the AOINIT IMOD and is controlled from the AO initialization panel.

Return Codes

0	Function completed normally.
4	For HVER/BVER, buffer transfer in progress.
8	Function not permitted in this environment.
16	Invalid function.

Sample Commands

```
address 'AO SAY CONSOLE'
address AO LOOP USER *

address 'AO SAY CONSOLE'     /* Send to console       */
say 'Send data to console'
address 'AO SAY BUFR'        /* Resume send to buffer */

cdone=0
do while cdone=0
        address 'AO SAY HVER'     /* Transfer/close HTML   */
        if rc<>4 then cdone=1     /* HTML request finished */
        else x=wait('1')          /* wait 1 and try again  */
end
exit
```

ADDRESS CARD

The ADDRESS CARD command allows you to read data from SYSIPT, and is only available from the BIM$RXBA utility.

Syntax

```
ADDRESS CARD
ADDRESS CARD .
ADDRESS CARD anything
```

Environment

This ADDRESS environment is only available for REXX IMODs run in batch via the BIM$RXBA program.

Operand

anything	Some data is required to actually read the SYSIPT data. If no data is supplied, the ADDRESS environment is set to CARD. When any unknown REXX statements are processed, they are passed to the ADDRESS card.

Return Codes

0	Function completed normally.
4	No SYSIPT data left to read.

Sample Commands

```
address CARD .
address CARD
```

Sample Program

```
rc=0;i=1
address card .
do forever until rc¬=4
   z.i=result;z.0=i
   i=i+1
   address card .
end
```

ADDRESS CICS

The ADDRESS CICS command allows you to pass valid JCLBCICS commands to JCLRCICS. The JCLRCICS task must be active in the target CICS partition.

Syntax

ADDRESS CICS *id function args*

Environment

This ADDRESS environment is only available if the BIM-FAQS/PCS JCLRCICS task is active in the target partition. Data is returned in the stem variable 'CICS.', where CICS.0 is the number of stems that are set.

Operands

id	Partition ID or DOS jobname.
function	CICS function to invoke. Valid functions are CEMT, OPEN, CLOSE, and START.
args	Optional args to pass to the CICS function.

Return Code

0	Function completed normally.

Sample Commands

```
address CICS F2 CEMT INQ
address CICS T1 CLOSE XYZ
```

Sample Program

```
pid='F2';cmd='CEMT INQ TAS'
address cics pid cmd
if datatype(cics.0)='N' then do
   do i = 1 to cics.0
       say cics.i
    end
end
```

ADDRESS CONSOLE

The ADDRESS CONSOLE command provides an interface to the VSE ASYNOC routine. Any valid AR or REPLY can be provided. POWER commands can also be supplied, but either the PWRCMD interface or the ADDRESS POWER interface is recommended. No information is returned.

Any valid REPLY or AR command up to 80 characters can be provided. PRTY REPLY will pass only 71 characters. For VSE/ESA 2+ systems the command length can be up to a maximum of 126 characters.

Return Codes

0	Function completed normally.
2	Function completed, but ASYNOC was busy and multiple attempts had to be made.
8	Severe error occurred. More information is available on the system console.

Sample Commands

```
address console
address console 'PRTY J'
address console '0 GO'
```

Sample Program

```
reply ='0 delete'
address console reply
if rc>=4 then say 'Unable to issue reply' reply
PID='BG'
reply ='delete'
address console
'PRTY REPLY' pid reply
if rc>=4 then say 'Unable to issue reply' reply
```

ADDRESS DISK

The ADDRESS DISK command allows you to read data from DISK, and is available only from the BIM$RXBA utility.

Syntax

```
ADDRESS DISK
ADDRESS DISK .
ADDRESS DISK anything
```

Environment

This ADDRESS environment is available only for REXX IMODs run in batch via the BIM$RXBA program. It reads records from the FILE GSPDSID and returns the records in the variable *RESULT*. It uses a DTFSD with fixed-length records of 121 bytes, and uses standard OPEN, READ, and CLOSE macros to read the file.

Operands

anything	Some data is required to actually read the GSPDSID file. If no data is supplied, the ADDRESS environment is set to DISK; as a result, when any unknown REXX statements are processed, they are passed to the address DISK.

Return Codes

0	Function completed normally.
4	No records left to read.

Sample Commands

```
address DISK .
address DISK
```

Sample Program

```
rc=0;i=1
address DISK
do forever until rc¬=0
   'READ'
   z.0=i
   z.i=result
   i=i+1
end
```

ADDRESS EPIC

The ADDRESS EPIC command provides an interface to the EPIC for VSE product.

Return Code

0	Function completed normally.

Sample Commands

ADDRESS EPIC EXTRACT RELEASE STEM *n.*

This form returns the current EPIC release as n.1 in the stem variables *n.1* through *n.i*, where *i* is the last DSN record number. The variable *n.0* is set to the maximum record number returned.

ADDRESS EPIC EXTRACT DSN ID *ii* STEM *n.* COUNT *c*

This form returns up to the first *c* dataset master records in the stem variables *n.1* through *n.i*, where *i* is the last DSN record number. The variable *n.0* is set to the maximum record number returned.

ADDRESS EPIC EXTRACT DSN ID *ii* STEM *n.* SKIP *s* COUNT *c*

This form returns up to *c* dataset master records in the stem variables *n.1* through *n.i*, where *i* is the last DSN record number. The first *s* records are skipped. The variable *n.0* is set to the maximum record number returned. Either or both of the SKIP and COUNT operands can be omitted.

ADDRESS EPIC EXTRACT DSN ID *ii* STEM *n*.

This form returns up to the first 100 dataset master records in the stem variables *n.1* through *n.i*, where *i* is the last DSN record number. The variable *n.0* is set to the maximum record number returned. This is a special form where both SKIP and COUNT are omitted.

ADDRESS EPIC EXTRACT VOL *vv* STEM *n*. SKIP *s* COUNT *c*

This form returns up to *c* volume records in the stem variables *n.1* through *n.i*, where *i* is the last volume record number. The first *s* records are skipped. The variable *n.0* is set to the maximum record number returned. Either or both of the SKIP and COUNT operands can be omitted.

ADDRESS EPIC EXTRACT DETAIL ID *ii* STEM *n*.

This form returns all detail records associated with DSN *ii* in the stem variables *n.1* through *n.i*, where *i* is the last record number. The variable *n.0* is set to the maximum record number returned.

ADDRESS EVENT

The ADDRESS EVENT command provides an interface to BIM-FAQS/PCS events.

Syntax

```
ADDRESS EVENT function eventname <argument>
```

Operands

function	Function can be DELETE, HOLD, POST, RESET, SATISFY, UNHOLD, or LIST. LIST returns the date in the stem variable EVENT., where EVENT.0 is the number of stems that are set.
eventname	Specify the event name to take action on.
argument	Argument is required for the SATISFY function. Argument can be any user-defined WHEN condition of AUX=, $JOB=, PROD=, VAR=, or EVENT=.

Return Codes

0	Function completed normally.
516	Event not found or invalid status.
520	Unimplemented feature.
524	No condition found to satisfy.
528	$FAQSAO not found.
532	System GETVIS failure.
536	Invalid command for BIM-FAQS/ASO.
540	Security violation.

Sample Commands

```
address EVENT DELETE DAILYJOB
address EVENT SATISFY XYZ '$JOB=ABC'
```

Sample Program

```
parse arg cmd
 address EVENT cmd
 if rc>=0 then say 'ADDRESS EVENT Failed rc=||rc
```

ADDRESS EXPLORE

The ADDRESS EXPLORE command provides an interface to CA-EXPLORE for VSE and CA-EXPLORE for CICS-VSE.

Syntax

```
ADDRESS EXPLORE prodid EXTRACT data qualifiers
                       CMD command
                       BUFFSIZE nnnnn
```

Operands

prodid

Prodid is one of the following product IDs: COMMON, CICS, or VSE.

COMMON EXTRACT data

When prodid is COMMON (that is, CA-EXPLORE is installed but not necessarily active), data can have one of the following values:

MACHINFO	Machine-specific information
LIBRLDIR	Library directory data
LIBRLFND	Finds a library member
LIBRLGET	Gets a library member

CICS EXTRACT data

When prodid is CICS (that is, CA-EXPLORE for CICS-VSE is active), data can be one of the following:

- JOBNAME_LIST
- INTERVAL_DATA
- PLOT_DATA
- PLOT_VARS

VSE EXTRACT data

When prodid is VSE (that is, EXPLORE for VSE is active), data can have one of the following values:

SAMPLE	Last 60 minutes of system status
SYSSTATE	Current system status
CURSYSI	Current SYSTIMEI system statistics
CURCHNI	Current SYSTIMEI channel statistics
CURDEVI	Current SYSTIMEI device statistics
PART	Current SYSTIMEI partition statistics
TASK	Current SYSTIMEI task statistics

TASKDISK	Task disk statistics
TASKDDSN	Task disk dataset statistics
TASKPGML	Task phase usage statistics
TASKOIO	Task other I/O statistics
DSNNAME	Get dataset name for area
SDSNNAME	Get dataset names for a disk

qualifiers

Qualifiers is one or more of the following positional keywords, used to further identify or limit the scope of the data to be extracted.

Important! The PID, TID, CUU, ID, and STEM keywords must appear in the specified sequence, if more than one is used. Only STEM is required.

PID id	Two-character partition ID, such as AR, F1, C1, or Z1.
TID tid	Task ID, specified as four hexadecimal digits (for example, 0021, 0033, 005B).
CHAN id	Channel ID, specified as two hexadecimal digits (for example, 00, 01, 0F).
DEVTYPE type	VSE device type. Can also be DISK, TAPE, OTHER, or a 2-hex-digit specification of a VSE device type. For a list of valid VSE device types, see the VSE MAPDEVTY macro.
CUU ccuu	Device address of up to four hexadecimal digits (for example, 0080, 0120, 022c).
ID id	Eight-character identifier, such as jobname or node ID.
STEM stemvar	Stem variable into which the results are placed. stemvar must be 30 characters or less, including the period. stemvar.0 will contain the count of stemvar.n's which were created. STEM is required.
optional parms	Additional parameters that further identify or limit the scope of the data to be extracted. Any other information that the particular extract requires, must be placed after the stemvar name.

CMD and BUFFERSIZE

CMD and *BUFFSIZE* pass the following values:

command	A particular CA-EXPLORE/VSE SBAT operator command, such as MAPDISK, CLOSE EVSEARC, or INDICATE SYSTEM.
nnnnn	*nnnnn* must be between 1024 and 32,768. If you receive a RC=4 (buffer not large enough to contain all data), increasing the buffer size may solve the problem.

Sample Program

```
/* Basic GREXX to access EXPLORE/VSE data thru  EXPLXDAT */
signal on error
address EXPLORE COMMON EXTRACT RELEASE
say "EXPLORE/COMMON release=" result
address EXPLORE VSE EXTRACT RELEASE
say "EXPLORE/VSE release=" result
address EXPLORE CICS EXTRACT RELEASE
say "EXPLORE/CICS release=" result
address EXPLORE VSE EXTRACT SYSSTATE STEM STUFF.
say "Records returned=" stuff.0
do i = 1 to stuff.0
  result = stuff.i
  say c2x(result)
end
return
ERROR:
 say "Unexpected return code" rc "from command:"
 say sourceline(sigl)
 say "at line" sigl
 exit
```

ADDRESS OUTPUT

The ADDRESS OUTPUT command allows you to send data to SYSLST, and is available only from the BIM$RXBA utility.

Syntax

ADDRESS OUTPUT ***printline***

Environment

This ADDRESS environment is available only for REXX IMODs run in batch via the BIM$RXBA program. Since a DTFDI is used, records sent have a maximum length of 120 bytes.

Operand

printline — *printline* is required, and consists of two parts. The first part is the first byte, which must be an ASA control character, such as the following:

blank	Space 1 line
0	Space 2 lines
-	Space 3 lines
+	Suppress space
1	Skip to line 1 on new page

The second part is the remaining 120 bytes, which are for data you wish to print.

Return Code

0	Function completed normally.

Sample Commands

```
address OUTPUT '1This is a new page'
```

Sample Program

```
cc='1'
do i=1 to 5
  address output cc||'This is line';
  cc=' '
end
```

ADDRESS PDATE

Address PDATE is used to verify a BIM-FAQS/PCS event day KEYWORD against the supplied or current date.

Operands

KEYWORD	This is the keyword to verify. The length of the keyword is from 1 to 8 characters. This must be the first parameter supplied and may be the only parm used.
DATE=	The date in (YY/MM/DD) format to use for the keyword verification.
HOL=	Which Holiday Id table to use when doing the comparison. Holiday Id 000 will be used if none is supplied
CYC=	Which Cycle Id table to use when doing the comparison. Cycle Id 000 will be used if none is supplied
ACT=	Any holiday action to be used with the keyword. Valid values are: S O P W N. If no value is supplied, the default of 'keyword is not valid on a holiday' is assumed
WCT=	Any non-workday action to use with the keyword. Valid values are: S O P W N. If no value is supplied, the default of 'keyword is valid on a non-workday' is assumed

Return Codes

0	Keyword is valid for the date specified.
4	Keyword is not valid for the date specified.

ADDRESS PDS

ADDRESS PDS is used to access a PDS member.

Environment

This ADDRESS environment is available only if the customer has BIM-FAQS/PCS or from the BIM$RXBA utility.

Operands

function	Specify either GET or PUT. GET causes the member to return in the PDS.stem variable. PDS.0 indicates the number of records returned. Five bytes of control information begin each record. If you modify a record, blank out the control information before issuing the PUT function. The appropriate control information is supplied if blanks or binary zeros appear in the first 5 bytes.
args	Supply the PDS name, member name, and type, using the format *pds:membname.typ*.

Return Code

0	Function completed normally.

Sample Command

```
address pds 'GET MON:JCLSCHED.CTL'
```

Sample Program

```
/* */
pds.=''
address pds 'GET MON:JCLSCHED.CTL'
if rc=0 then do
   do i=1 to pds.0
      if substr(pds.i,11,7)=='&FAQSAO' then do
          x='     '||substr(pds.i,6,5)
          x=x||'$FAQSXX'||substr(pds.i,18)
          pds.i=x
      end
   end
   address pds 'PUT MON:JCLSCHED.CTL'
   say 'JCLSCHED.CTL successfully updated'
   exit
end
exit rc
```

ADDRESS POWER

The ADDRESS POWER command provides an interface to POWER to issue POWER commands through the VSE ASYNOC routine. ADDRESS POWER also checks to see if the POWER command processor is busy before issuing the command to ASYNOC. Any valid POWER command can be provided. It is recommended that POWER commands be issued through the PWRCMD function if you have either BIM-FAQS/ASO or BIM-FAQS/PCS installed. No information is returned.

Any valid POWER command up to 80 characters can be provided. For VSE/ESA 2+ systems the command length can be up to a maximum of 126 characters.

Return Codes

0	Function completed normally.
2	Function completed, but ASYNOC or POWER was busy and multiple attempts had to be made.
8	VSE asynchronous operator task or POWER is busy. Twenty attempts at 2-second intervals were made to perform the command, but ASYNOC or POWER was busy each time.
16	Severe error occurred. More information is available on the system console.

Sample Commands

```
address POWER
address POWER 'D Q'
address POWER 'L LST,ALL,CRDATE=05/04/92'
```

Sample Program

```
cmd ='D Q'
address power cmd
if rc>=4 then say 'Unable to issue reply' reply
d = date(b)
d = d-5
d = date(U,d,'b')
address power
'L LST,ALL,CRDATE<='||d
'PRTY REPLY' pid reply
if rc>=4 then say 'Unable to issue reply' reply
```

ADDRESS PROGRAM

This ADDRESS environment allows you to call programs from REXX IMODs and return data via the RESULT variable. The LOAD or CALL option causes the requested program to be loaded if not already available, and then executed. The FREE option allows you to remove the program from the partition, thereby releasing the GETVIS associated with it. The called program is provided the following:

Input Registers

R1	address of the passed parameters (parms).
R13	Address of a standard 78 byte save area.
R14	Return address.
R15	Address of program loaded.

And then is responsible to return the following:

Output Registers

R0	length of results from program, if any.
R1	address of results from program, up to a maximum of 20000 characters.
R15	Return code.

Note that if you return a result, the address containing this result must be in defined storage (ex. Not Freevised) until the result has been acknowledged by REXX code (the RESULT REXX variable has been set). Also keep in mind that the program must be reentrant and will be entered in AMODE=31. If the program does not meet the above criteria, it is responsible for preventing reentrancy issues and/or setting up access in AMODE=24 and then returning with AMODE=31. You could write a driver program which calls other programs and then restores the original environment before returning to REXX.

Environment

This ADDRESS environment is available for BIM-FAQS/ASO and BIM-FAQS/PCS users.

Operands

name	Name of the program to be executed.
parms	Parameters to pass to the program up to a maximum of 128 characters in length.

Return Codes

0	Function completed normally.
32	Load of requested program failed.
n	Value of register 15 upon return from the program.

Sample Commands

```
parmlst = 'This is a parameter list'
address program 'CALL TESTPROG' parmlst
address program 'FREE TESTPROG'
```

ADDRESS SCHEDULE

The ADDRESS SCHEDULE command provides an interface to JCLSCHED.

Operand

cmd	xxx

Return Codes

0	Function completed normally.
16	Severe error occurred. More information is available on the system console.

Sample Command

```
address SCHEDULE '$TEST'
```

Sample Program

```
cmd ='schedule'
address schedule cmd
if rc>=0 then say 'SCHEDULE FAILED rc=||rc
```

ADDRESS SYS

The ADDRESS SYS command provides an interface to the VSE AR (ASYNOC routine). ADDRESS SYS is similar to ADDRESS CONSOLE except that ADDRESS SYS allows a BIM-FAQS/ASO-defined command to be issued. This allows you to start an IMOD from an IMOD, which can cause a *recursive* loop. Use this in special cases only, since uncontrolled loops can cause system failure.

Any valid AR, REPLY, or BIM-FAQS/ASO-defined console command can be provided. POWER commands can also be supplied, but either the PWRCMD interface or the ADDRESS POWER interface is recommended.

No data is returned.

Operand

cmd	Any valid REPLY or AR command up to 80 characters. For VSE/ESA 2+ systems the command length can be up to a maximum of 126 characters.

Return Codes

0	Function completed normally.
2	Function completed, but ASYNOC was busy and multiple attempts had to be made.
8	VSE asynchronous operator task busy. Twenty attempts at 2-second intervals were made to perform the command, but ASYNOC was busy each time.
16	Severe error occurred. More information is available on the system console.

Sample Commands

```
address sys 'cicsrepl csmt...'
address sys
```

Sample Program

```
cmd ='start'   /* start defined as console command/imod */
address SYS cmd
if rc>=4 then say 'Unable to start the start IMOD'
```

Chapter 5
REXX Functions

BIM REXX employs both general REXX functions and product-specific functions. Functions comprise a series of instructions that can receive and process data, then return a value to the IMOD that issued the function.

Functions differ from instructions because a function performs a process and returns the result of the process to REXX. An instruction is a single-step process.

ABBREV(pattern,string,length)

Purpose

The ABBREV function returns 1 if *string* is an abbreviation of *pattern*. Otherwise, it returns 0.

Operands

pattern	Full-length string. This is the pattern that is checked for an abbreviation.
string	String to check against *pattern*. ABBREV checks to see whether the string is an abbreviation of the pattern.
length	Optional minimal length *string* needed for a match. This is used if you want a minimum abbreviation.

For example, you may want to check for abbreviations of the patterns 'STARTUP' and 'SHUTDOWN'. Since 'S' is ambiguous, you need a minimum abbreviation of 2 ('ST' and 'SH').

Examples

```
ABBREV('TEST','TE',1)   ==>  1
ABBREV('TEST','TE',3)   ==>  0
ABBREV('TEST','TES',3)  ==>  1
ABBREV('TEST','TX')     ==>  0
```

The ABBREV function simplifies command checking in the following sample.

Sample Program

```
parse upper arg cmd data
select
   when abbrev('STARTUP',cmd,2) then ...
   when abbrev('SHUTDOWN',cmd,2) then ...
   otherwise ...
end
```

ABS(number)

Purpose

ABS returns the absolute value of *number* and is formatted according to the NUMERIC settings.

Operands

number	Decimal number.

Examples

```
ABS('-101.1')          ==>  101.1
ABS('101.12345678)     ==>  101.123457
ABS('-32')             ==>  32
```

Sample Program

```
if ¬datatype(z,'N') then  x=0
                    else x=ABS(z-4)
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a ABS( ) function on a nonnumeric string. The IMOD terminates and the following message displays:

GRX0041I Error within nnnnnnnn, line xx Bad arithmetic conversion

To avoid a runtime error, you should use the DATATYPE function to ensure that the string is a numeric. See the sample above.

ADDRESS( )

Purpose

ADDRESS returns the name of the current environment. This value is for information purposes only, since you cannot specify the operand of the ADDRESS instruction as a variable.

Operands

( )	No operands are required.

Examples

```
                            Required product
ADDRESS AO                        ASO PCS
ADDRESS CARD
ADDRESS CICS                          PCS
ADDRESS CONSOLE
ADDRESS DISK
ADDRESS EPIC                         EPIC
ADDRESS EVENT                         PCS
ADDRESS EVSE                   CA-EXPLORE
ADDRESS OUTPUT
ADDRESS PCL                           PCS
ADDRESS PCS                           PCS
ADDRESS PDS                           PCS
ADDRESS PDATE                         PCS
ADDRESS POWER                     ASO PCS
ADDRESS PROGRAM                   ASO PCS
ADDRESS PUNCH
ADDRESS SYS                       ASO PCS
ADDRESS VERSION                       PCS
```

ARG (<n<,option>>)

Purpose

If no operand is coded, ARG returns the number of arguments. If an operand is specified, the value of that argument is returned (null if the specified argument number is not present). REXX IMODs called from commands can have only 1 or 0 arguments. Procedures can have *n* arguments.

Operands

n	Optional decimal argument number to check.
option	Option to test for the existence of the *n*th arg.

E	Returns 1 if nth arg exists. Otherwise, it returns 0.
O	Returns 1 if nth arg is omitted. Otherwise, it returns 0.

Examples

```
Example called with no args passed
* CURSOR
ARG()                   ==>  0
ARG(2)                  ==>  ''
ARG(2,'e')              ==>  0
ARG(1,'0')              ==>  1


Example called via call imod a,,b
* CURSOR
ARG()                   ==>  3
ARG(1)                  ==>  'value of a'
ARG(2)                  ==>  ''
ARG(2,'e')              ==>  0
ARG(1,'0')              ==>  0
```

Sample Program

```
a='1 2'
call test a,'test'
exit result
/* ---------------- */
/* Sample procedure */
/* ---------------- */
test: procedure
if arg(2,'e') then return '-1'
parse arg a b
if a < b then return a
         else return b
```

ASOENV ( )

Purpose

The ASOENV command provides an interface to the FAQSAO IMOD processor to determine in what environment the IMOD was initiated.

This function returns information on the IMOD execution environment (for example, ASO or PCS) and information on the operating system environment (for example, VSE/SP or VSE/ESA). It also provides information about whether the IMOD was called as a result of an SMSG, AR command, message action, or GEM.

The data passed as a result of a $MSG action is in fixed format and cannot be parsed with blank-delimited words.

Environment

This function is only available for an IMOD running through the FAQSAO processor.

Operands

( )	No operands are required.

Sample Program

The following example shows how data is parsed and returned using an X=ASOENV() call. For a more complete example, and to test data passed, see the $ARG IMOD supplied at installation.

```
x=asoenv()                 /*  get environment          */
parse var x env lvl avl imod pds type data
select
   when type='$CMD' then cmd=data
   when type='SMSG' then user=data
   when type='$MSG' then do
        action=substr(data,1,12)
        pid=substr(data,13,2)
        jobname=substr(data,15,8)
        phasename=substr(data,23,8)
        time=substr(data,31,8)
   end
   otherwise nop
end
say 'ENVIRONMENT' env
say 'IMOD       ' imod
say 'user       ' user
say 'cmd        ' cmd
say 'action     ' action
say 'pid        ' pid
say 'jobname    ' jobname
say 'phasename  ' phasename
say 'time       ' time

---------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
ENVIRONMENT ASO
IMOD         ARG
user
cmd
action      MSGOP
pid         F7
jobname     JCLSCHED
phasename   JCLSCHED
time        17:18:04
```

BITAND(string1,string2,pad)

Purpose

BITAND returns the result of the Boolean AND of *string1* and *string2*. The shorter of the two strings is padded with *pad*. If *pad* is omitted, the shortest string determines the number of positions that participate in the AND operation. If *string2* is omitted, *pad* is ANDed with each *string1* position.

Operands

string1	First operand to AND.
string2	Optional second operand to AND.
pad	Optional pad character if *string1* and *string2* are not the same length.

Examples

```
BITAND('Y','Y')             ==>  'Y'
BITAND(' ','Y')             ==>  '40'x
BITAND('0101'x,'0101'x)     ==>  '0101'x
BITAND('0101'x,'0101'x)     ==>  '0101'x
BITAND('0101'x,'0011'x)     ==>  '0001'x
BITAND('0101',,'ff'x)       ==>  '0101'x
```

Sample Program

```
parse arg cmd
if   BITAND(cmd,,'bf'x)=cmd
   then say cmd 'is all lowercase letters'
   else say cmd 'is not all lowercase'
```

BITOR(string1,string2,pad)

Purpose

BITOR returns the result of the Boolean OR of *string1* and *string2*. The shorter of the two strings is padded with *pad*. If *pad* is omitted, the shortest string determines the number of positions that participate in the OR operation. If *string2* is omitted, *pad* is ORed with each *string1* position.

Operands

string1	First operand to OR.
string2	Optional second operand to OR.
pad	Optional pad character if *string1* and *string2* are not the same length.

Examples

```
BITOR('ab','4040'x)         ==>  AB
BITOR('Bob',,' ')           ==>  BOB
BITOR('01'x,'4040'x)        ==>  '4140'x
BITOR('01','F0F0'x,'F0')    ==>  01
BITOR('ab','FF'x,'40')      ==>  'FFC2'x
```

Sample Program

```
parse arg cmd
if   BITOR(cmd,,' ')=cmd
   then say cmd 'is all UPPERCASE letters'
   else say cmd 'is not all UPPERCASE'
```

BITXOR(string1,string2,pad)

Purpose

BITXOR returns the result of an exclusive OR of *string1* and *string2*. If *pad* is omitted, the shortest string determines the number of positions that participate in the XOR operation. If *string2* is omitted, *pad* is XORed with each *string1* position.

Operands

string1	First operand to XOR.
string2	Optional second operand to XOR.
pad	Optional pad character if *string1* and *string2* are not the same length.

Examples

```
BITXOR('Y','Y')              ==>  '00'
BITXOR(' ','Y')              ==>  'y'
BITXOR('0101'x,'0101'x)      ==>  '0000'x
BITXOR('0101'x,'0011'x)      ==>  '0110'x
BITXOR('0101'x,,'ff'x)       ==>  'FEFE'x
```

Sample Program

```
toggle: procedure expose x
x=BITXOR(x,,'ff'x)
return
```

B2C(binary-string)

Purpose

The B2C function converts a binary string of 0's and 1's to a character string. Intervening blanks in the binary string are removed.

Operands

binary-string	Can be any-length string of 0's and 1's. If there are not multiples of 4 binary digits, the 0's are added to the left of the string. Blanks can be imbedded in the string, but only at 4-digit boundaries.

Examples

```
B2C('110000001')              ==>  A
B2C('1111000111110000')       ==>  10
B2C('1111 0001 1111 0000')    ==>  10
```

Sample Program

```
x=b2c('11000001')
y=b2c('1111000111110000')
say x y b2c('1111 0001 1111 0000')
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a B2C( ) function on a non-binary string. The IMOD terminates and the following message displays:

```
GRX0015I Error in nnnnnnnn, line x Invalid Hexadecimal or Binary string
```

To ensure that the string is a binary string and thereby avoid a runtime error, use the DATATYPE function.

B2X(binary-string)

Purpose

The B2X function converts a binary string of 0's and 1's to a string of hexadecimal characters. The returned string is in uppercase, consisting of 0-F characters and no intervening blanks.

Operands

binary-string	Can be any-length string of 0's and 1's. If there are not multiples of 4 binary digits, the 0's are added to the left of the string. Blanks can be imbedded in the string, but only at 4-digit boundaries.

Examples

```
B2X('111')                ==>  07
B2X('1110001')            ==>  71
B2X('11100001')           ==>  E1
B2X('11 0000 1111')       ==>  30F
```

Sample Program

```
x=X2D(b2x('111'))
y=X2C(b2x('11100001'))
z=x2b(b2x('11 0000 1111'))
say x y z

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
7 A 001100001111
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a B2X( ) function on a non-binary string. The IMOD terminates and the following message displays:

GRX0015I Error in nnnnnnnn, line x Invalid Hexadecimal or Binary string

To ensure that the string is a binary string and thereby avoid a runtime error, use the DATATYPE function.

CENTER(string,length<,pad>)

Purpose

The CENTER function returns *string* centered within *length* positions padded on the left and right with the *pad* character.

Operands

string	String to be centered.
length	Length of output to center string within.
pad	Optional pad character if *string* is smaller than *length*.

Examples

```
CENTER('header',10)        ==>  '  header  '
CENTER('header',10,'-')    ==>  '--header--'
CENTER('header',4,'-')     ==>  'eade'
```

Sample Program

```
say CENTER('Now is',10)
say CENTER('the time',10)
say CENTER('for',10)
say x y z

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
  Now is
 the time
   for
```

CENTRE(string,length<,pad>)

Purpose

The CENTRE function returns *string* centered within *length* positions padded on the left and right with the *pad* character.

Operands

string	String to be centered.
length	Length of output to center string within.
pad	Optional pad character if *string* is smaller than *length*.

Examples

```
CENTRE('header',10)         ==>  '  header  '
CENTRE('header',10,'-')     ==>  '--header--'
CENTRE('header',4,'-')      ==>  'eade'
```

Sample Program

```
say CENTRE('Now is',10)
say CENTRE('the time',10)
say CENTRE('for',10)
say x y z

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
  Now is
 the time
   for
```

CLUSTER(parms)

Purpose

To list VSAM clusters from an IMOD.

Operands

parms	Depends on function selected.

Examples

Contact BIM-FAQS/ASO technical support if you wish to use this function.

COMPARE(string1,string2,pad)

Purpose

The COMPARE function returns 0 if the strings match. Otherwise, it returns the position of the first character that does not match.

Operands

string1	String to be centered.
string2	Length of output to center string within.
pad	Optional pad character if one *string* is smaller than the other. The default pad character is blank.

Examples

```
COMPARE('xyz','xyz')        ==>  0
COMPARE('xyz.','xyz','.')   ==>  0
COMPARE('xyz','xYz')        ==>  2
COMPARE('1234','123')       ==>  4
```

Sample Program

```
x=readcons('is this it?')
if compare(x,'This is it') = 0
   then say 'ok'
   else say 'String' x 'incorrect'

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
ok
```

COPIES(string,n)

Purpose

The COPIES function returns a string made up of n copies of the input string. No intervening spaces are added and one copy of the string is placed adjacent to the next copy.

Operands

string	String to be copied.
n	Number of copies to make.

Examples

```
COPIES('+...','3')        ==>  +...+...+...
COPIES('xyz','1')         ==>  xyz
COPIES('xyz','0')         ==>  ''
```

CP(cmd,ASIS)

Purpose

The CP command provides an interface to VM. The command is issued using a VM diagnosis.

Data is returned in a stem variable.

A maximum of 80 data lines can be returned. Stem.0 contains the number of stem variables returned. If the CP reply buffer is too small to handle all the data returned, the first stem variable is set to CP REPLY BUFFER EXCEEDED.

Operands

cmd	Any valid CP command for which the VSE machine has the proper VM CLASS authority.
ASIS	ASIS can be coded to allow the data to be issued without uppercasing it. This allows SMSG hex data to be sent to other machines, but you must ensure the actual commands are passed in the proper case.

Return Codes

0	CP command issued. Check the *stem* variable for results.
8	No command provided.
-3	No valid BIM-FAQS/ASO product code.

Sample Program

```
z.=''
z.=cp('ind')
if rc=0 then do
            do i=1 to z.0
               say strip(z.i)
                end
            end
            else do
                 say 'CP failed rc='||rc
            end

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
AVGPROC-014% 01
STORAGE-031% PAGING-0003/SEC STEAL-000%
Q0-00001(00000)                          DORMANT-00057
Q1-00001(00000)           E1-00000(00000)
Q2-00000(00000) EXPAN-001 E2-00000(00000)
Q3-00002(00000) EXPAN-001 E3-00000(00000)
PROC 0001-014%
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a CP( ) function and not assigning stem data. The IMOD terminates and the following message displays:

GRX0108I Error in nnnnnnnn, line x Function returned unassigned stem data

CPUID( )

Purpose

The CPUID command provides CPUID and VM machine name information. On non-VM machines, only the CPUID is returned.

Operand

( )	No operands are required.

Sample Program

```
say 'The current CPU ID is' word(cpuid(),1)
say 'The current MACHINE is' word(cpuid(),2)

-------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
The current CPU ID is FF17301330900000
The current MACHINE is DEVVSE
```

C2D(string<,n>)

Purpose

C2D returns the decimal value of the characters given as input. The result must not have a value greater than the current NUMERIC DIGITS.

Operands

string	String to convert to decimal.
n	If n is not specified, the result is assumed to be an unsigned number.

Examples

```
C2D('0F'x)      ==> 15
C2D('3')        ==> 243
C2D('F3'x)      ==> 243
C2D('0081'x)    ==> 129
C2D('ABC')      ==> 12698307
C2D('ABCde')    ==> 8.32196280E+11
C2D('fffb')     ==> 65531
C2D('fffb',2)   ==> -5
```

C2X(string)

Purpose

C2X returns the hexadecimal value of the characters given as input.

Operand

string	String to convert to display hexadecimal.

Examples

```
C2X('3')      ==> F3
C2X('AB')     ==> C1C2
```

DATATYPE(string<,type>)

Purpose

DATATYPE returns a code for the type of data in string. It returns 'NUM' for decimal or 'CHAR'. If the second operand is specified, DATATYPE returns 1 if the desired type was found; otherwise, it returns 0.

Operands

string	String to determine datatype of.	
type	Optional desired type character to check for specified type.	
	A	‘Alphanumeric' returns 1 if string contains only characters in the range('a','z'), range('A','Z'), and range('0','9').
	B	‘Binary' returns 1 if string contains only characters '0' or'1's.
	L	'Lowercase' returns 1 if string contains only lowercase letters.
	M	'MixedCase' returns 1 if string contains only lowercase and uppercase letters.
	N	'Numeric' returns 1 if string contains only numbers in the range(0,9).

S	'Symbol' returns 1 if string contains only characters that are REXX symbols.
U	'UPPERCASE' returns 1 if string contains only uppercase letters.
W	'Whole-number' returns 1 if string is a whole number under the current setting of NUMERIC DIGITS.
X	'heXadecimal' returns 1 if string is a valid hexadecimal number. Case is ignored. A null string is a valid hexadecimal number. Blanks are allowed as two-character pairs.

Examples

```
DATATYPE('     1  ')     ==>  'NUM'
DATATYPE('     X1 ')     ==>  'CHAR'
DATATYPE('1A')           ==>  'CHAR'
DATATYPE('1A','x')       ==>  1
DATATYPE('31f7','X')     ==>  1
DATATYPE('31f7','B')     ==>  0
DATATYPE('Bob','M')      ==>  1
DATATYPE('Bob','L')      ==>  0
```

Sample Program

```
if ¬datatype(z,'N') then  x=0
                    else  x=ABS(z-4)
```

DATE(<option<,date<,'B'>>>)

Purpose

If *date* is omitted, DATE returns the current date. The optional supplied date must be seven characters in *yyyyddd* format. If *option* is not specified, the format *dd Mmm yyyy* is used. For example, DATE() returns '20 Jul 1992' for July 20, 1992.

Operands

option	Optional format of date to return. Only the first character of 'option' is checked. This allows you to have a more descriptive option in your program or simply a letter.

B	Number of complete days since January 1, 0001 (Base days)
C	Number of days since January 1 of the start of the century.
D	Number of days so far this year, counting today.
E	Date in the format dd/mm/yy (European format).
J	Date in the format yyddd (Julian or OS format).
M	Name of the current month (examples 'August', 'October').
O	Date in the format yy/mm/dd (Ordered format).
S	Date in the format yyyymmdd (Sorted format).
U	Date in the format mm/dd/yy (USA format).
W	Weekday (e.g., 'Tuesday', 'sunday').

date	Optional date to convert to desired format. Must be in the format yyyyddd unless B is specified for base days.
'B'	Date provided is in base days. This option is useful for doing calculations on dates without having to worry about leap years and crossing boundaries.

Examples

```
DATE()                   ==>  '20 Jul 1992'
DATE(B)                  ==>  727398
DATE(C)                  ==>  33804
DATE(D)                  ==>  202
DATE(E)                  ==>  '20/07/92'
DATE(J)                  ==>  92202
DATE(M)                  ==>  'July'
DATE(O)                  ==>  '92/07/20'
DATE(S)                  ==>  '19920720'
DATE(U)                  ==>  '07/20/92'
DATE(W)                  ==>  'Monday'
DATE(U,'1992001')        ==>  '01/01/92'
DATE(U,'727398','b')     ==>  '07/19/92'
```

Sample Program

```
d=date('b')                 /* get base days   */
d=d-5                       /* subtract 5 days */
d=date(U,d,'b')             /* get new date    */
address power 'L LST,ALL,CRDATE<='||d
```

DELSTR(string,n<length>)

Purpose

The DELSTR function returns the string resulting when the substring starting at *n* and of length *length* is removed from *string*. DELSTR removes the designated substring, concatenates the leading and trailing substrings, and returns the result.

Operands

string	String to alter.
n	Starting position.
length	Number of positions to remove.

Examples

```
DELSTR('123456',3)       ==>  '12'      <==  LEFT('123456',2)
DELSTR('123456',3,2)     ==>  '1256'
DELSTR('123456',7)       ==>  '123456'
DELSTR('12345',1,4)      ==>  '5'       <==  RIGHT('12345',1)
```

DELWORD(string,n,length)

Purpose

The DELWORD function returns the string resulting when the word(s) starting at word *n* are removed from *string* (*length* words are removed). DELWORD removes the designated words, concatenates the leading and trailing substrings, and returns the result.

Operands

string	String to alter.
n	Starting word number.
length	Optional number of words to remove.

Examples

```
DELWORD('REXX is not fun',3,1)    ==>  'REXX is fun'
DELWORD('REXX is not fun',3)      ==>  'REXX is'
DELWORD('REXX is not fun',1,3)    ==>  'fun'
```

DIGITS( )

Purpose

The DIGITS function returns the value of NUMERIC DIGITS.

Operand

( )	No operands are required.

Example

```
DIGITS()           ==>  9
```

Sample Program

```
say digits()
numeric digits 5
say digits()

-------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
9
5
```

D2C(number<,n>)

Purpose

The D2C function returns the character string that is the binary of *number*.

Operands

number	Decimal number to convert to character.
n	Length of result. If *number* is negative, *n* must be specified and must be large enough to contain the result.

Examples

```
D2C(256)          ==>  '100'x
D2C(255,2)        ==>  '00FF'x
D2C(4095)         ==>  '0FFF'x
D2C(4096)         ==>  '1000'x
D2C(4096,1)       ==>  '00'x
D2C(-1,2)         ==>  'ffff'x
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a D2C( ) function and *number* is not a decimal whole number. The IMOD terminates and the following message occurs:

GRX0041I Error within nnnnnnnn, line xx Bad arithmetic conversion

D2X(number<,n>)

Purpose

The D2X function returns the hexadecimal string that is the value of *number*.

Operands

number	Decimal number to convert to hexadecimal.
n	Optional length.

Examples

```
D2X(256)          ==>  '100'
D2X(256,2)        ==>  '00'
D2X(4095)         ==>  'fff'
D2X(4096)         ==>  '1000'
D2X(4096,1)       ==>  '0'
D2X(-1,2)         ==>  'ff'
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a D2X( ) function and *number* is not a decimal whole number. The IMOD terminates and the following message appears:

GRX0041I Error within nnnnnnnn, line xx Bad arithmetic conversion

ERRORTEXT(n)

Purpose

The ERRORTEXT function returns the REXX error message associated with the error number specified. n must be a number in the range of 0-99. If there is no error message defined for the number, '** undefined' is returned.

Operand

n	Decimal number between 0 and 99.

Example

```
ERRORTEXT(41)      ==> Bad arithmetic conversion
```

Sample Program

```
do i=1 to 99
  say errortext(i)
end

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
** undefined
** undefined
Program is unreadable
Program interrupted
        .
        .
        .
```

FIND(string,tgt)

Purpose

The FIND function returns 0 if *tgt* not found in *string*. Otherwise, FIND matches blank-delimited words, is insensitive to the number of blanks preceding or following a word, and returns the word number.

Operands

string	String to search for *tgt*.
tgt	Search argument (series of words).

Examples

```
FIND(' now   is   the    time','the time')  ==>  '3'
FIND('How now brown cow','now cow')         ==>  '0'
```

FORM( )

Purpose

The FORM instruction returns the value of NUMERIC FORM.

Operand

( )	No operands required.

Example

```
FORM()          ==>  'SCIENTIFIC'
```

Sample Program

```
say 'The current form is' form()
numeric form engineering
say 'The current form is'||form()

-------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
The current form is SCIENTIFIC
The current form is ENGINEERING
```

FORMAT(number,<integer>,<decimal>)

Purpose

The FORMAT function formats and rounds the specified number. If *number* only is specified, the operation is equivalent to doing number=number+0.

Operands

integer	The number of characters to be used in the integer part of the number. If integer is too small to contain the number, an error results.
decimal	The number of characters to be used in the decimal part of the number.

Examples

```
Format('10',6)       ==>  '    10'
Format('10.52',4,1)  ==>  '  10.5'
Format('10.52',3,2)  ==>  ' 10.52'
Format('10.52',2,2)  ==>  '10.520'
```

Sample Program

```
/* BIM$RXBA exec to read product and prices then format */
/* into dollars and cents.                              */
address card
total=0;rc=0
do i= 1 to 1000 until rc¬=0
   'READ'
   z.i=result; z.0=i; total=total+word(result,2)
end
address output
cc=' '
call header
do i=1 to z.0
   if i // 60 = 0 then call header
x=format(word(z.i,2),,2)
   x= right(x)
   cc||substr(word(z.i,1),1,10) '$'||x
end
   cc||'            -------'
   cc|'Total:     ' right(format(total,,2),6)
exit
/* HEADER is a PROCEDURE to localize variable cc */
header: procedure
cc='1'
address output cc||'Product:     Price:'
return

--------------------------------------------------------------

// EXEC BIM$RXBA,SIZE=BIM$RXBA,PARM='FORMAT'
BIKE   79
BAT    15.95
BALL   .79
/*

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
Product:     Price:
BIKE         79.00
BAT          15.95
BALL          0.79
BALL          0.79
            -------
Total:       96.53
```

FUZZ( )

Purpose

The FUZZ function returns the value of NUME FUZZ.

Operand

( )	No operands required.

Example

```
FUZZ()          ==>  0
```

GETVIS(pid)

Purpose

The GETVIS command provides GETVIS information about partitions or the SVA. GETVIS returns the following information:

- Length of GETVIS control information
- Maximum GETVIS used to this point for the current jobstep
- Free space
- Used space
- Maximum block of contiguous free GETVIS

Operand

pid	Any valid partition ID: BG, F1-FB, AR for the SVA, or dynamics.

Return Codes

0	Function completed normally.
8	Invalid parameter length. Must be 2-characters long.
16	Partition ID not found.
-3	No valid BIM-FAQS/ASO product code

Example

```
GETVIS('BG') ==> '00000016 00001268 00000986 00001030 00000801'
```

Sample Program

```
say '   Length    Max       Free      Used      Max'
say '   Getvis    Used      Getvis    Getvis    Block'
pid= pidlist('B') ||'AR'   /* set possible pids             */
j= length(pid) % 2        /* calculate # of Pids in string */
do i = 0 to j-1             /* loop for number of pids-1     */
   y=substr(pid,i*2+1,2)    /* calculate index to pid string */
   x=getvis(y)             /* get getvis for pid y          */
   if rc=0 then call getdspl x
           else say 'Getvis failed rc='||rc 'pid='||y
end
exit
/***************************************************************/
/*  getdspl: get dispaly information.                          */
/***************************************************************/
getdspl:
parse var x ctllen maxused free used contig
total=free+used
total=substr(strip(total,l,'0')||'K',1,9)
maxused=substr(strip(maxused,l,'0')||'K',1,9)
free=substr(strip(free,l,'0')||'K',1,9)
used=substr(strip(used,l,'0')||'K',1,9)
contig=substr(strip(contig,l,'0')||'K',1,9)
if total='K' then say 'Getvis Area for' y 'Not Initialized'
             else say y total maxused free used contig
return

-------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
   LENGTH    MAX       FREE      USED      MAX
   GETVIS    USED      GETVIS    GETVIS    BLOCK
F1 344K      256K      171K      173K      131K
F3 3100K     2740K     413K      2687K     377K
F7 2012K     728K      1400K     612K      1296K
F5 2020K     1064K     1186K     834K      1053K
F4 2048K     964K      1133K     915K      1085K
F2 1944K     756K      1236K     708K      1201K
FB 3508K     2904K     788K      2720K     788K
GETVIS AREA FOR BG NOT INITIALIZED
GETVIS AREA FOR FA NOT INITIALIZED
F8 912K      852K      132K      780K      61K
F9 1916K     1816K     428K      1488K     154K
GETVIS AREA FOR F6 NOT INITIALIZED
AR 1476K     800K      870K      606K      722K
```

INDEX(haystack,needle,start)

Purpose

INDEX returns 0 if string *needle* does not occur within string *haystack*. Otherwise, it returns the decimal character position.

Operands

haystack	String to search.
needle	Desired string.
start	Optional decimal starting position.

Examples

```
INDEX('ABCDEFGH','A')      ==>  '1'
INDEX('ABCDEFGH','a')      ==>  '0'
INDEX('ABCDEFGH','F')      ==>  '6'
INDEX('ABCDEFGH','I')      ==>  '0'
INDEX('ABCDEFGH','CDE')    ==>  '3'
INDEX('ABCDEFGH','CDE',4)  ==>  '0'
```

Sample Program

```
string='Now is the time'
data='not'
say result
exit

test: procedure string data
n=index(string,' ')
n=index(string,' ',n)
string=insert(data||' ',string,n)
return string

--------------------------------------------------------------

* CURSOR
1...5...10...15...20...25...30...35...40...45...50...55...60..
Now is not the time
```

INSERT(new,target,<n>,<length>,<pad>)

Purpose

Insert a new string into a target string at the specified position. If a length is specified, the *new* string is truncated or padded on the right with the *pad* character.

Operands

new	New string to insert.
target	Target string.
n	Insert new beginning after the nth character (default is 0).
length	Length of new string to insert (default is length of new).
pad	Pad character (default is blank).

Examples

```
INSERT('Now is the','time',,11,' ')    ==>  'Now is the time'
INSERT('is','Now  the time',4)         ==>  'Now is the time'
```

Sample Program

```
string='Now is the time'
data='not'
say result
exit

test: procedure string data
n=index(string,' ')
n=index(string,' ',n)
string=insert(data,string,n,length(data)+1,' ')
return string
```

```
1...5...10...15...20...25...30...35...40...45...50...55...60..
Now is not the time
```

JOBACCT(pid)

Purpose

The JOBACCT command provides job accounting information about jobs running in the system. JOBACCT works like ASO J / PRTY J. The following accounting information is returned:

- Jobname
- Phase name
- Job duration
- Step duration
- CPU seconds
- SIO count

Operand

pid	Any valid partition ID: BG, F1-FB, or dynamics.

Return Codes

0	Function completed normally.
8	Invalid parameter length. Must be two characters long.
16	Partition ID not found.
-3	No valid BIM-FAQS/ASO product code.

Sample Program

```
x='Job       Phase     Job       Step         Cpu        SIO'
say '   '||x
x='Name      Name      Duration  Duration     Seconds    Cnt'
say '   '||x
pid='F2'
x=jobacct(pid)      /* get getvis for specified pid  */
if rc=0 then say pid x
        else say 'Jobname failed rc='||rc 'pid='||pid


1...5...10...15...20...25...30...35...40...45...50...55...60..
   Job       Phase     Job       Step         Cpu        SIO
   Name      Name      Duration  Duration     Seconds    Cnt
F2 CICSMROA  DFHSIP    08.45.35  08.41.46     021.93     10177
```

JOBACCT('CPU') / JOBACCT('PAG')

Purpose

This command provides either the CPU or PAGing statistics for the past ten minutes, in one minute intervals.

Return Codes

0	Function completed normally.
8	Invalid parameter length. Must be three characters.
-3	No valid BIM-FAQS/ASO product code.

Sample Program

See the sample IMOD $GETCPU for more information.

JOBNAME(pid)

Purpose

The JOBNAME command retrieves the jobname from the COMREG for the specified partition ID. If a partition ID is not entered, the jobname is the message that triggered the IMOD or the jobname where FAQSAO resides.

Operand

pid	Any valid partition ID: BG, F1-FB, or dynamics.

Return Codes

0	Function completed normally.
8	Invalid parameter length. Must be two characters long.
16	Partition ID not found.
-3	No valid BIM-FAQS/ASO product code.

Example

```
JOBNAME('BG')        ==>  'NONAME '
JOBNAME('F2')        ==>  'CICSA  '
```

Sample Program

```
x=jobname('BG')
if rc=0 then say strip(x) 'is in BG'
        else say 'Jobname failed rc='||rc
x=jobname('1')
if rc=0 then say x
        else say 'Jobname failed rc='||rc

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
LIBR is in BG
Jobname failed rc=8
```

JUSTIFY(string,length,pad)

Purpose

The JUSTIFY function returns the string resulting when the individual word of *string* is justified with character *pad* to a width of *length* positions. Justification proceeds left to right. To perform justification right to left as in most SCRIPT processors, use the REVERSE function. The online help panels use this extensively.

Operand

string	String to be justified.
length	Width to justify within.
pad	Optional pad character (default is blank).

Examples

```
JUSTIFY('Now it the time',3)     ==>  'Now'
JUSTIFY('Now it the time',16)    ==>  'Now  is the time'
JUSTIFY('Now it the time',17)    ==>  'Now  is  the time'
JUSTIFY('Now it the time',18)    ==>  'Now  is  the  time'
```

Sample Program

```
work='Now is the time for all'
say justify(work,24)
work=reverse(justify(reverse(work),24))
say work

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
Now  is  the time for all
Now is the time  for  all
```

LASTPOS(needle,haystack,start)

Purpose

The LASTPOS function returns the decimal value of the last occurrence of *needle* within the string *haystack*. Returns 0 if not found.

Operands

needle	Search argument
haystack	String to search
start	Optional starting position for backward search.

Examples

```
LASTPOS(' ','123 xyz abc')       ==>  8
LASTPOS(' ','123 xyz abc',8)     ==>  8
LASTPOS(' ','123 xyz abc',7)     ==>  4
LASTPOS('qrs','123 xyz abc')     ==>  0
```

Sample Program

```
parm='e d c b a'
index = length(parm)
do i = 1 to words(parm)
   index=lastpos(' ',parm,index-1)
  say subsr(parm,index+1,1)
   index=index-1
end

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
a
b
c
d
e
```

LEFT(string,length,pad)

Purpose

The LEFT function returns *string* left justified to a width of *length*.

Operands

string	String to left justify.
length	Width of resulting string.
pad	Optional pad value (default is blank).

Examples

```
LEFT('abcd',6)       ==>  'abcd  '
LEFT('1234',8,'.')   ==>  '1234....'
LEFT('1234',2,'.')   ==>  '12'
```

Sample Program

```
linelen=25
call lformat 'Widget','78'
call lformat 'Steal Bar','100'
call lformat 'Program Errors','7'
exit

lformat: procedure expose linelen
   say left(arg(1),linelen,'.')   right(arg(2),4)
  return

-------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
Widget...................   78
Steal Bar................  100
Program Errors...........    7
```

LENGTH(string)

Purpose

The LENGTH function returns the length of *string*, and returns 0 for a null string.

Operand

string	String to check length

Examples

```
LENGTH(000012) returns 2.
LENGTH('abcd') returns 4.
LENGTH('') returns 0.
```

Sample Program

```
a='54a92'
n=0
c=0
do i = 1 to length(a)
  if datatype(substr(a,1,i),'N') then n=n+1
                                  else c=c+1
end
say 'There are' n 'numeric digits in the string'
say 'There are' c 'non-numeric digits in the string'

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
There are 4 numeric digits in the string
There are 1 non-numeric digits in the string
```

LIBR(function,parms)

Purpose

To access VSE library members from an IMOD.

Operand

function	LIBLIST, LIBINFO, SUBLIST, MEMLIST, or MEMBER
parms	Depends on function selected

Examples

Contact BIM-FAQS/ASO technical support if you wish to use this function.

LINESIZE( )

Purpose

The LINESIZE function returns the decimal value of the line size to be used for the SAY instruction.

Operand

( )	No operands required.

Example

```
LINESIZE()       ==>  80
```

Sample Program

```
parm='This is a test of the linesize() function'
if length(parm) <= linesize()
   then say  parm
   else do
        substr(parm,1,linesize())
        substr(parm,linesize(),80)
   end
```

LISTCAT(parms)

Purpose

To list VSAM clusters and get information about total space free, used, etc. from an IMOD.

Operand

parms	Depends on function selected

Examples

Contact BIM-FAQS/ASO technical support if you wish to use this function.

MAX(number,number,...)

Purpose

The MAX function returns the maximum of the whole number operands. Up to ten decimal operands can be passed.

Operand

number	Whole number

Examples

```
MAX(1,2,20,30,4,5,6,7,8)   ==>  30
MAX(1,2,3,4,5,6,7,8,9,10)  ==>  10
MAX(-2,-33,-14)            ==>  -2
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a MAX( ) function and number is not a decimal. The IMOD terminates and the following message occurs:

GRX0041I Error within nnnnnnnn, line xx Bad arithmetic conversion

MESSAGE(<pid>,<count>,<scan>,<start>)

Purpose

The MESSAGE command retrieves messages from the hardcopy file. A maximum of 200 messages can be returned. Messages are returned in LIFO format. The most recent messages are first. A maximum of 2000 messages can be read (VSE/ESA releases prior to 2), or, for VSE/ESA release 2 and above, to either the previous IPL or the start of the hardcopy file, whichever is reached first.

Operand

pid	Any valid partition ID: BG, F1-FB, AR, or dynamics. If a partition ID is not specified, all messages are returned.
count	The number of messages to return.
scan	Optional scan information. If scan data is specified, the requested messages must contain the scan data.
start	Optional number of messages to skip before data is returned.

Return Codes

0	Function completed normally.
8	Function completed normally.
-3	No valid BIM-FAQS/ASO product code.

Sample Program

```
/* take a terminal out of service and get the node  */
/* Note: a real example is provided in $CYCLE       */
arg pid term
call $cicsrep pid 'CEMT S TER('||term||') OUT'
cics. = message(pid,'8')
do i=1 to cics.0
  if word(cics.i,3)='Ter('||term||')' then do
      j=i-1
     parse var cics.j . 'Net(' node ')'
     if node='' then do
         j =i-2
        parse var cics.j . 'Net(' node ')'
      end
     leave
   end
end
node= strip(node)
say 'NODE='||node 'is available'

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
Node=D72L304 is available
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a MESSAGE( ) function and not assigning stem data. The IMOD terminates and the following message appears:

GRX0108I Error in nnnnnnnn, line x Function returned unassigned stem data

MIN(number,number,...)

Purpose

The MIN function returns the minimum of the whole number operands. Up to ten whole number operands can be passed.

Operand

number	Whole number

Examples

```
MIN(1,2,20,30,4,5,6,7,8)    ==>  1
MIN(55,97,33,1111,10,99)    ==>  10
MIN(1,2,3,4,5,6,7,8,9,10)  ==>  1
MIN(-2,-33,-14)             ==>  -33
```

MSG(mid<,start>)

Purpose

The MSG command looks up BIM-FAQS/ASO, BIM-FAQS-PCS, GSS, IBM, or user-defined message explanations. A maximum of 200 lines of message description can be returned. If more than 200 lines are needed, you can use the start parameter to read more data. There is an IMOD ($MSG) provided on installation to display message descriptions on the system console. This allows operators to display message descriptions and actions on the system console without referring to a manual. Specific messages can also trigger full explanations automatically.

Data is returned in a stem variable. Stem.0 contains the number of stem variables returned.

Operands

mid	Any message or help ID defined in the IBM IESMSGS VSAM file or in the FAQSMSG VSAM file (VSE/ESA prior to release 2), or the IBM online messages file (VSE/ESA release 2 and above).
start	Optional start line to read large messages such as VSAMOPEN.

Return Codes

0	Function completed normally.
8	Invalid parameter length. Must be 1-9 characters long.
16	Message not found.
-3	No valid BIM-FAQS/ASO product code.

Sample Program

```
z.=''
z.=msg('4444D')
if rc=0 then do
            do i=1 to z.0
               say strip(z.i,t)
             end
        end
        else do
             say 'MSG failed rc='||rc
        end

--------------------------------------------------------------
```

```
* CURSOR
1...5...10...15...20...25...30...35...40...45...50...55...60..
4n44T      OVERLAP ON UNEXPRD FILE (filename)(SYSxxx = cuu)

Explanation: Refer to Figure 8 in IBM VSE/SP Messages and
Codes.  It gives additional explanations regarding the message
identifier and system action.
An extent specified in an EXTENT statement would overlap
one extent of an unexpired non-VSAM-managed file on the volume
named in the message.
 System Action:
For type code I - The system cancels the job.
For type code D - The system waits for an operator response.

Programmer Response: Compare the high and low extent limits
specified in the EXTENT statement (or your latest LSERV output,
       .
       .
       .
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a MSG() function and not assigning stem data. The IMOD terminates and the following message appears:

GRX0108I Error in nnnnnnnn, line x Function returned unassigned stem data

OVERLAY(new,tgt,<n>,<len>,<pad>)

Purpose

The OVERLAY function returns the result of overlaying *tgt* starting at position *n* for length *len* with the string *new*.

Operands

new	String to overlay *tgt* with.
tgt	String to be altered.
n	Starting position of *tgt* to overlay.
len	Number of positions of *tgt* to overlay.
pad	Optional pad character (default is space).

Examples

```
OVERLAY('X','ABCD',2,1)            ==>  'AXCD'
OVERLAY('X','ABCD',2,2,' ')        ==>  'AX D'
OVERLAY('APPLE','ABCD',1,6,'.')    ==>  'APPLE.'
OVERLAY('APPLE','ABCD',1,1,' ')    ==>  'ABCD'
```

PHASE(pid)

Purpose

The PHASE command retrieves the phase name from the COMREG for the specified partition ID. This is an 8-character name returned, padded on the right with blanks.

Operand

pid	Any valid partition ID: BG, F1-FB, or dynamic partition.

Return Codes

0	Function completed normally.
8	Invalid parameter length. Must be two characters long.
16	Partition ID not found.
-3	No valid BIM-FAQS/ASO product code.

Sample Program

```
x=phase('BG')
if rc=0 then say x
        else say 'Phase failed rc='||rc
x=phase('1')
if rc=0 then say x
        else say 'Phase failed rc='||rc
if phase('F2') ='DTSINIT ' then saTC'
                           else say 'MSG F2'

-------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
LIBR
Phase failed rc=8TC
```

PIDLIST(<type>)

Purpose

The PIDLIST command retrieves a list of static partitions and any active dynamic partitions.

Operand

type	Defaults to *S* for a list of static partitions. *B* can be specified for a list of static and currently active dynamic partitions. *D* can be specified for a list of dynamic partitions only. If * is used, the current partition is returned.

Return Codes

0	Function completed normally.
8	Invalid parameter.
-3	No valid BIM-FAQS/ASO product code.

Examples

```
PIDLIST('B')  ==> 'F1F3F2FAF9Y3F8F7F6F5F4BGT2T1'
PIDLIST('D')  ==> 'Y3T2T1'
PIDLIST('B')  ==> 'F1F3F2FAF9F8F7F6F5F4BG'
PIDLIST()     ==> 'F1F3F2FAF9F8F7F6F5F4BG'
```

Sample Program

```
pid=pidlist('B')
j= length(pid) % 2
do i = 0 to j-1
   y=substr(pid,i*2+1,2)
   x= jobname(y)
   if rc=0 then do
if x='NO NAME' then nop
else jobname=jobname||strip(x)||','
     end
end
   say strip(jobname,'b',',')
exit

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
IPWPOWER,DTSINIT,JCLSCHED,BATCH44
```

POS(needle,haystack,start)

Purpose

The POS function returns position where *needle* is first found within *haystack*. Returns 0 if *needle* does not occur at all within *haystack*.

Operands

needle	String to search *haystack* for.
haystack	String to search
start	First position within *haystack* to check (scan start).

Examples

```
POS('F2','BG,F1,F2')    ==>  7
POS('FA','BG,F1,F2')    ==>  0
POS('abc','abcde')      ==>  1
```

Sample Program

```
/*$bump  bump a partitions priority */
 arg pid
 address console 'PRTY'
 p.=message('AR',8)
 do i = 1 to p.0
    if word(p.i,3)='PRTY' then leave
 end
 prty=word(p.i,4)
 say prty
 curloc=pos(pid,prty)
 if curloc=0 then return
 if curloc>= length(prty)-2 then do
   say pid 'is already the highest priority'
   return
 end
 if curloc>=length(prty)-5 then do
    say pid 'is already the second highest priority'
    return
 end
 newprty= substr(prty,1,curloc-2)
 newprty=newprty||substr(prty,curloc+1,length(prty)-1-curloc)
 newloc=index(newprty,',',curloc)
 prty=substr(newprty,1,newloc)||pid||','
 prty=prty||substr(newprty,newloc+1,length(newprty)-newloc)
 address console 'PRTY' prty
 say prty
 return

---------------------------------------------------------------
 1...5...10...15...20...25...30...35...40...45...50...55...60..
 F9=F6=BG=F5,F7,FA,FB,F4,F8,F3,F2,F1
 F9=F6=F5,BG,F7,FA,FB,F4,F8,F3,F2,F1
```

POST(event)

Purpose

The POST command posts an BIM-FAQS/PCS event. The event must be in the current schedule.

Operand

event	Any current event from 1-8 characters.

Return Codes

0	Function completed normally.
2	Event already complete or not found.
8	Invalid parameter length. Must be 1-8 characters.

Sample Program

```
x=post('FAQSAO')
if rc=0 then say 'Event' event 'posted'
        else say 'POST failed rc='||rc

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
Event FAQSAO posted
```

POWER(<queue>,<jobname>,<class>)

Purpose

The POWER command retrieves POWER queue information. XPCC communicates to POWER to retrieve the requested data.

Data is returned in a stem variable. Stem.0 contains the number of stem variables returned.

Operand

queue	Any valid POWER queue (RDR, LST, PUN, or XMT). The default is RDR.

jobname	Eight-character jobname, or eight characters with a preceding * to denote generic. The default is all jobs.
class	Class that is displayed. The default is all classes.

Return Codes

0	Function completed normally.
8	Invalid jobname parameter. Must be 8 characters or less.
-3	No valid BIM-FAQS/ASO product code.

Sample Program

```
z.=power('LST,,'A')
if rc=0 then do
            do i=1 to z.0
               say z.i
             end
        end
        else do
             say 'POWER failed rc='||rc
        end

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
 1-8    Jobname    (8)          39-42    FNO         (4)
  9                               43
10-14   Jobnumber  (5)          44-51    Date        (8)
  15                              52
16-18   Job suffix (3)          53-60    User (to)   (8)
  19                              61
  20    Priority   (1)          62-69    Node (to)   (8)
  21                              70
  22   Disposition (1)          71-78    Node (from) (8)
  23                              79
  24   Class       (1)          80-87    User (from) (8)
  25                              88
  26   SYSID       (1)          89-104   Userinfo    (16)
  27
28-33  pages       (6)
  34
35-37  copies      (3)
  38
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a POWER( ) function and not assigning stem data. The IMOD terminates and the following message appears:

GRX0108I Error in nnnnnnnn, line x Function returned unassigned stem data

PWRCMD(cmd)

Purpose

The PWRCMD command issues the POWER command through XPCC and returns messages. XPCC tells POWER to retrieve the requested data.

Data is returned in a stem variable. Stem.0 contains the number of stem variables returned.

Operand

cmd	Any PDISPLAY, PALTER, PHOLD, PDELETE, PRELEASE, or PINQUIRE. Other commands are valid (if authorized), such as PBROADCAST, PGO, PSTOP, and PSTART.

Return Codes

0	Function completed normally.
8	No command provided.
-3	No valid BIM-FAQS/ASO product code.

Sample Program

```
z.=pwrcmd('D LST')  /*  issue PWRCMD and return data in stem */
if rc=0 then do     /*  check for non-zero return code       */
  do i=1 to z.0
     say strip(z.i)
   end              /*  end do loop                          */
end                 /*  end if                               */
pcmd='L LST,XXXX'   /*  set a variable                       */
say 'AO '||pcmd
z.=pwrcmd(pcmd)     /* issue pwrcmd and return results       */
say strip(z.1)

---------------------------------------------------------------

* cursor
1...5...10...15...20...25...30...35...40...45...50...55...60..
1R46I    LIST QUEUE   P D C S  PAGES  CC FORM
1R46I  FAOXPCC  27343 4 H A        3   1
1R46I  COMPLK   27359 3 D A        7   1      TO=(CAM2)
1R46I  GSJOBCTL 27360 3 D A        3   1      TO=(KJM)
1R46I  FAQSVMCF 27361 3 D A        3   1      TO=(KJM)
1R46I  FAQSINIT 27362 3 D A        3   1      TO=(KJM)
1R46I  AUDPRTV2 27314 4 L B        3   1      FROM=(JCLXPCC)
1R46I  CATALR   27325 4 H B        1   1      TO=(BCP2)
AO L LST,XXXX
1R88I  NOTHING TO DELETE
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a PWRCMD( ) function and not assigning stem data. The IMOD terminates and the following message appears:

GRX0108I Error in nnnnnnnn, line x Function returned unassigned stem data

QUEUED( )

Purpose

The QUEUED function returns number of items on the program stack (0 if none). PUSH adds entries to the stack.

RANDOM(min,max,seed)

Purpose

The RANDOM function returns a random number between min and max. If min and max are both omitted, a random integer is returned.

Operand

min	Optional minimum value to return.
max	Optional maximum value to return.
seed	Optional random number generator seed value.

If only one value is specified, it is assumed to be the *max* value, and zero is used for the *min* value.

A *seed* can optionally be used to start the random number generator off at a fixed point, with a reproducible sequence of values. This allows you to generate a pseudo-random set of numbers for testing purposes. By providing the *seed*, you always produce the same sequence of numbers.

All values must be non-negative.

Examples

```
RANDOM(1,10,5555)   ==>  5
RANDOM(10,20)       ==>  17
RANDOM(5555)        ==>  378
RANDOM(5555)        ==>  4001
```

Sample Program

```
 call pseudo_random 99
 call pseudo_random 10
 exit
 /* produce 10 pseudo random numbers always the same */
pseudo_random: procedure
 arg x
 y=random(1,10,x)      /* set pre-determined seed  */
 list=''
 do i = 0 to 10
    list=list random(1,10)    /* produce list        */
 end
 say list
 return

---------------------------------------------------------------

 1...5...10...15...20...25...30...35...40...45...50...55...60..
 7 1 10 1 10 6 5 3 3 10
 6 2 4 7 9 2 9 4 7 6
```

READCONS(<data>)

Purpose

The READCONS command issues a read to the system console. The reply ID is associated with the FAQSAO task that issues the read. A maximum of 100 characters can be read. The IMOD waits until the read is satisfied. If data is not specified, a default message is issued.

Operand

data	Option data to output.

Return Codes

0	Function completed normally.
-3	No valid BIM-FAQS/ASO product code.

Sample Program

```
/* called via console command PRTY intercept */
 arg cmd
 if words(cmd)=1 then address sys
 user= strip(userid())
 select
   when user='DEVVSE' then do
        /* message masking must be active to hide */
        /* on system console when user enters PW= */
       pw=readcons('enter password PW=XXXXXX')
       parse upper var pw 'PW=' pw
       if pw='OPERATIONS' then address sys cmd
                          else say 'SORRY Charlie'
        end
   when user='BOBSM'  then  address sys cmd
   when user='MERROW' then  address sys cmd
   otherwise say 'Sorry' user 'not allow to alter prty'
 end

---------------------------------------------------------------

 1...5...10...15...20...25...30...35...40...45...50...55...60..
 SORRY Charlie
```

REPLID(pid)

Purpose

The REPLID command retrieves outstanding replies for the system or specified partition ID. If a partition ID is not entered, all outstanding replies are returned.

Operand

pid	Any valid partition ID: BG, F1-FB, AR or dynamics. If a partition ID is not specified, all replies are returned.

Return Codes

0	Function completed normally.
8	Invalid parameter length. Must be two characters long.
16	Partition ID not found.
-3	No valid BIM-FAQS/ASO product code.

Examples

For VSE/ESA prior to release 2:

```
REPLID()            ==>  'BG-000 F2-002 F2-037'
REPLID('F2')        ==>  'F2-002 F2-037'
```

For VSE/ESA release 2 and above:

```
REPLID()            ==>  'BG-0000 F2-0002 F2-0037'
```

Sample Program

```
x=replid('F7')
if rc=0 then do
   if x=' ' then say 'There are no replies outstanding for F7'
            else say x
   end
   else do
        say 'Replid failed rc='||rc
   end
end
x=replid()
if rc=0 then do
   if x=' ' then say 'There are no replies outstanding'
            else say x
   end
   else do
        say 'Replid failed rc='||rc
   end
end
```

```
--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
There are no replies outstanding for F7
 BG-000 F3-003
```

For more examples, see the IMODs $CICSREP, $REPLY, and $REPLID, supplied at installation time.

REVERSE(string)

Purpose

The REVERSE function returns *string* reversed. The *string* is swapped end for end. For example, REVERSE('ABCD') returns 'DCBA'. To perform justification right to left as in most SCRIPT processors, the online help panels for REXX use the REVERSE function twice as demonstrated below.

Examples

```
REVERSE('ABCD')       ==>  'DCBA'
REVERSE('123456789')  ==>  '989654321'
```

Sample Program

```
work='Now is the time for all'
work=reverse(justify(reverse(work),24))
say work

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
Now is the time  for  all
```

RIGHT(string,len,pad)

Purpose

The RIGHT function returns *string* right justified to *len* positions.

Operand

string	String to right-justify.
len	Width to right-justify string within.
pad	Optional pad character (default is blank).

Examples

```
RIGHT('abcd',6)       ==>   '  abcd'
RIGHT('1234',8,'.')   ==>   '....1234'
RIGHT('1234',2)       ==>    '34'
```

Sample Program

```
linelen=25
call rformat 'Widget','78'
call rformat 'Steal Bar','100'
call rformat 'Program Errors','7'
exit

rformat: procedure expose linelen
   say left(arg(1),linelen,'.') right(arg(2),4)
  return

-------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
Widget...................   78
Steal Bar................  100
Program Errors...........    7
```

SESSION(args)

Purpose

To establish and utilize a BIM-FAQS/ASO session from within a REXX IMOD. It is strongly recommended that you use the provided $SESSION IMOD if you want to exercise this function. See the comments at the start of the $SESSION IMOD for further details. You can also look at the $SESTEST, $SESTST1, and BIMTCPFQ IMODs to see the $SESSION IMOD in use.

Examples

```
SESSION(userid,cmd,'I')
SESSION(token,cmd)
SESSION(token,cmd,'E')
```

SIGN(number)

Purpose

The SIGN function returns -1 if number<0, 0 if number=0, 1 if number>0.

Operand

number	Decimal value

Examples

```
SIGN('1.1')    ==>   1
SIGN('0')      ==>   0
SIGN('0.0')    ==>   0
SIGN('-0')     ==>   0
SIGN('- 1')    ==>  -1
SIGN(-2.1)     ==>  -1
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a SIGN( ) function and the number is not decimal. The IMOD terminates and the following message displays:

GRX0041I Error within nnnnnnnn, line xx Bad arithmetic conversion

SOURCELINE(<n>)

Purpose

The SOURCELINE function returns the number of lines in the current IMOD, or returns the nth line.

Operand

n	Option line number of IMOD to return. If n is not specified, the number of lines in the IMOD is returned.

Examples

```
SOURCELINE()         ==>   76
SOURCELINE(5)        ==>   '  do i = 1 to n'
```

Sample Program

```
/* sourceline example */
say 'There are' sourceline() 'lines in this imod'
do i = 1 to sourceline()
  say sourceline(i)
end

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
There are 5 lines in this imod
/* sourceline example */
say 'There are' sourceline() 'lines in this imod'
do i = 1 to sourceline()
   say sourceline(i)
end
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a SIGN() function and number is not a decimal. The IMOD terminates and the following message appears:

GRX0041I Error within nnnnnnnn, line xx Bad arithmetic conversion

SPACE(string,n,pad)

Purpose

The SPACE function returns the string resulting when each of the words of *string* is appended to the next word with *n pad* characters between words. Any leading or trailing blanks are first removed from *string*.

Operands

string	String to format.
n	Optional number of *pad* characters between words of string. The default is 1.
pad	Optional *pad* character (default is blank).

Examples

```
SPACE(' one  two three four  ',1)  ==>  'one two three four'
SPACE('1 2 3 4',4,'.')             ==>  '1....2....3....4'
SPACE('1 2 3 4',0)                 ==>  '1234'
```

Sample Program

```
x='1 2 3 4'
x=SPACE(x,4,'.')
say x
x=translate(x,' ','.')
say x
x=SPACE(x,0)
say x

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
1....2....3....4
1    2    3    4
1234
```

STATUS(pid)

Purpose

The STATUS command retrieves the runcode or TSS byte for each task in a specified partition. The TSS is explained in the online message facility. Type *MSG TSS* from a FAQS console for an explanation of the TSS.

Operands

pid	Any valid partition ID: BG, F1-FB, or AR. Any active dynamic partition.

Return Codes

0	Function completed normally.
8	Invalid parameter length. Must be two characters long.
16	Partition ID not found.

Examples

```
status('BG')  ==>   21-82
status('F1')  ==>   22-83 35-82
```

Sample Program

```
pid=pidlist('B')||'AR'     /* set possible pids             */
j=length(pid)%2            /* calculate # of Pids in string */
do i = 0 to j-1            /* loop for number of pids-1     */
   say substr(pid,i*2+1,2) status(substr(pid,i*2+1,2))
end

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
F1 22=82 33=82 35=82 34=82 36=82
F6 27=82 4A=82
BG 21=82
F3 24=82 3B=82 3D=82 3E=82 48=82
F5 26=82 32=82 30=82 31=82
F7 28=82 37=83 38=82 39=82 3A=82 3C=82
F4 25=82
F2 23=82 40=82 41=82 42=82 44=82 45=82 46=82 43=82
F8 29=82
FA 2B=82
FB 2C=82
F9 2A=82 3F=82 47=82
AR 01=80 02=80 03=80 06=80 08=80 07=80 09=80 0A=80 0C=80 0F=80
```

STRIP(string,option,char)

Purpose

The STRIP function returns string resulting when leading/trailing or both leading and trailing *char* characters are removed from *string*.

Operands

string	String to be stripped.
option	Optional specifier (L or T or B, default is B).
char	Optional character to strip (default is blank).

Examples

```
STRIP(' ABC ')         ==>  'ABC'
STRIP(' ABC ','B')     ==>  'ABC'
STRIP(' ABC ','T')     ==>  ' ABC'
STRIP(' ABC ','L')     ==>  'ABC '
STRIP('0785','L','0')  ==>  '785'
```

SUBSTR(string,start<,end><pad>)

Purpose

SUBSTR returns the defined substring. If the defined substring is partially outside of the specified string, the result is padded on the right with the *pad* character (or spaces if *pad* is omitted). You may also want to look at the LEFT and RIGHT functions, or use the PARSE VAR with template.

Operands

string	String to work on.
start	Decimal starting position.
end	Decimal length. If not specified, the rest of the string is assumed.
pad	Pad character if we exceed *string* boundary. The default for *pad* is a blank.

Examples

```
SUBSTR(,1,8,'_')              ==>  '________'
SUBSTR('12345678',4,2)        ==>  '45'
SUBSTR('12345678',8,2)        ==>  '8 '
SUBSTR('12345678',8,2,'-')    ==>  '8-'
SUBSTR('12345678',20,2)       ==>  '  '
SUBSTR('12345678',5)          ==>  '5678'
SUBSTR('12345678 ',5)         ==>  '5678 '
```

SUBWORD(string,n<,length>)

Purpose

The SUBWORD function returns substring of *string* from word *n* of *length* words. The string does not have any leading or trailing blanks.

Operands

string	String to operate upon.
n	Starting word number.
length	Number of words. Defaults the rest of the words in the string.

Examples

```
SUBWORD('one two three',2,1)     ==>  'two'
SUBWORD('one two   three',2,2)   ==>  'two   three'
SUBWORD('one two three',2)       ==>  'two three'
```

SYMBOL(symbol)

Purpose

The SYMBOL command returns the state of the named symbol. If the specified symbol in not valid, BAD is returned. If *symbol* is a name of a variable that has a value, VAR is returned. Otherwise, LIT is returned.

Operand

symbol	Any valid REXX symbol

Examples

```
x='1'
symbol(x)     ==>  'VAR'
symbol('x')   ==>  'LIT'
symbol('+')   ==>  'BAD'
```

TIME(option)

Purpose

TIME returns the current time. If option is not specified, the format *hh:mm:ss* is used. For example, TIME( ) returns '14:30:00' at 2:30 in the afternoon. Only the first capitalized character of the option is checked.

Operand

option Optional format of time (default is hh:mm:ss).

C	Civilian format h:minutes<am|pm>
E	Elapsed time as ssssssss.uuuuuu (seconds.microseconds).
H	Number of hours since midnight.
L	Time in format hh:mm:ss.uuuuuu (uuuuuu=microseconds).
M	Number of minutes since midnight.
R	Elapsed time as ssssssss.uuuuuu. Resets elapsed time
S	Number of seconds since midnight.

The elapsed time clock is started (at zero) for the first call. It accumulates time thereafter until a TIME('R') call is made.

Examples

```
TIME()     ==>   '21:33:16'
TIME('R')  ==>   0
TIME('C')  ==>   '9:33pm'
TIME('H')  ==>   21
TIME('L')  ==>   '21:33:16.722401'
TIME('M')  ==>   1293
TIME('N')  ==>   '21:33:16'
TIME('S')  ==>   77596
TIME('E')  ==>   0.000904
```

TRANSLATE(string,tblo,tbli,pad)

Purpose

The TRANSLATE function returns *string* after translation is performed. If both *tblo* and *tbli* are omitted, TRANSLATE converts *string* to all uppercase. When *tblo* and *tbli* are specified, each occurrence of the first character in *tbli* in string is replaced by the first *tblo* character.

Operand

string	String to translate.
tblo	Optional output character table.
tbli	Optional input character table.
pad	Optional pad character (default is blank). Is added to the *tblo* table.

Examples

```
TRANSLATE('abc')                          ==>   'ABC'
TRANSLATE('abc',' ','b')                  ==>   'a c'
TRANSLATE('abc','123456789','abc')        ==>   '123'
TRANSLATE('abcd ','123456789','abc')      ==>   '123d'
TRANSLATE('abcd ','123','abcd')           ==>   '123 '
TRANSLATE('abcd ','123','abcd','.')       ==>   '123.'
```

TRUNC(number<,n>)

Purpose

The number is truncated to the specified decimal places, or trailing zeros are added.

Operand

number	Number to truncate..
n	Number of decimal places. The default is 0.

Examples

```
TRUNC(122.7)       ==>   122
TRUNC(3.14578,2)   ==>   3.14
TRUNC(2.6,2)       ==>   2.60
TRUNC(10,2)        ==>   10.00
```

USERID( )

Purpose

The USERID function returns the user identifier. This allows you to implement security on any IMOD based upon the user who is performing a command. The user identifier is set depending on how the IMOD is run.

Environment	**User ID Setting**
Batch	PCSBAT or BIM$RXBA
JCL	JOBCNTRL
CMS	PCSBAT
Console CMD	VM MACHINE name or CPUID
SMSG to VSE	VM MACHINE name that did SMSG
ONLINE	BIM-FAQS/ASO USERID

Operand

( )	No operands are required.

VALUE(name<,newvalue>)

Purpose

The VALUE function returns the value of the symbol *name*, and optionally allows you to set a new value at the same time. The VALUE() function allows you to dynamically build variables and assign and retrieve their values. This allows you to replace some of the functionality of the INTERPRET instruction. INTERPRET is not implemented due to BIM's implementation of compiled REXX.

Operands

name	Symbol name to return value of.
value	New value to assign to the VALUE of name.

Sample Program

```
/* sample value */
bob='9';test='bob';i=3;bob3='test'
say value(bob)
say value(test)
say value('test')
say value(i)
say value(test||i)
say value(test||'3')
say value(test,newvalue)
say value(test)
say value('test')
say value('bob')

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
 9
 9
 bob
 3
 test
 test
 9
 newvalue
 bob
 newvalue
```

VERIFY(string,ref<,MatchNomatch>,<start>)

Purpose

The VERIFY function returns 0 if *string* only contains characters in list *ref*. Otherwise, it returns first position that is not in *ref*.

Operands

string	String to scan for only *ref* characters.
ref	List of characters to check string against.
match	Optional match flag, Match or Nomatch. Nomatch is the default. If Match is specified, VERIFY returns 0 or the first character position in string that is found in *ref*.
start	Optional position in string to start at (default is 1).

Examples

```
VERIFY('123',xrange(0,9))     ==>   0
VERIFY('1239',xrange(0,8))    ==>   4
VERIFY('abc','ABC')           ==>   1
VERIFY('abc','abC','M')       ==>   1
VERIFY('abc','abC','N')       ==>   3
VERIFY('abc','123','M')       ==>   0
VERIFY('abc','123','n')       ==>   1
```

VSAM(function, parms)

Purpose

To read or write to VSAM clusters from an IMOD.

Operand

function	READ or WRITE
parms	Depends on function selected

Examples

Contact BIM-FAQS/ASO technical support if you wish to use this function.

VSSPACE(parms)

Purpose

To display space information about a VSAM catalog from an IMOD.

Operand

parms	Depends on function selected

Examples

Contact BIM-FAQS/ASO technical support if you wish to use this function.

VTAM('cmd',<time>)

Purpose

The VTAM command allows you to issue and receive VTAM commands and messages through a secondary programmable operator interface. The FAQSVSPO interface needs to run as a subtask in the FAQSAO partition. This command is only available for IMODs processed by the FAQSAO task.

Data is returned in a stem variable. Stem.0 contains the number of stem variables returned.

Operands

cmd	VTAM command to issue. However, VTAM cannot be terminated from this interface.
time	Time interval in which VTAM messages are received. Default is two seconds.

Return Codes

0	VTAM command issued. Check the *stem* variable for results.
8	No *cmd* provided.
32	The FAQSVSPO task not active in same partition as FAQSAO.
-3	No valid BIM-FAQS/ASO product code.

Sample Program

```
/* Note: a real example is provided in $CYCLE       */
z.=vtam(d net,act,id='BIM$TIDR')
do i=1 to z.0
   say z.i
end
```

```
---------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
D NET,ACT,ID=BIM$TIDR
IST097I  DISPLAY  ACCEPTED
IST075I  NAME = BIM$TIDR         , TYPE = APPL
IST486I  STATUS= CONCT     , DESIRED STATE= CONCT
IST861I  MODETAB=***NA*** USSTAB=***NA*** LOGTAB=***NA***
IST934I  DLOGMOD=***NA***
IST597I  CAPABILITY-PLU INHIBITED,SLU INHIBITED,SESSION LIMIT
IST213I  ACBNAME FOR ID = BIM$TIDR
IST654I  I/O TRACE = OFF, BUFFER TRACE = OFF
IST171I  ACTIVE SESSIONS = 0000000000, SESSION REQUESTS = 000
```

```
IST172I  NO SESSIONS       ACTIVE
IST314I  END
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a VTAM( ) function and not assigning stem data. The IMOD terminates and the following message appears:

GRX0108I Error in nnnnnnnn, line x Function returned unassigned stem data

Another possible error is that the FAQSVSPO task is not in the same partition as FAQSAO.

WAIT(sec)

Purpose

The WAIT command waits the specified number of seconds.

Operand

sec — Number of seconds to wait. For example, X=WAIT(3) specifies a wait of three seconds.

Return Code

0 — Function completed normally.

Sample Program

```
do forever
    x=wait('43200')              /* wait 1 day      */
    d= date('b')                 /* get base days   */
    d=d-5                         /* subtract 5 days */
    d=date(U,d,'b')               /* get new date    */
    z.=pwrcmd('l lst,crdate<='||d)
end
```

WORD(string,n)

Purpose

The WORD function returns word number *n* from *string*. Returns null if no word *n* in *string*.

Operands

string	String to scan for word *n*
n	Word number

Examples

```
WORD(' one  two three four  ',1)  ==>  'one'
WORD(' one  two three four  ',2)  ==>  'two'
WORD(' one  two three four  ',3)  ==>  'three'
WORD(' one  two three four',4)    ==>  'four'
WORD(' one  two three four  ',5)  ==>  ''
```

Sample Program

```
test='BIM-FAQS/ASO BIM-FAQS/PCS BIM-GSS'
do i = 1 to words(test)
  say word(test,i)
end

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
BIM-FAQS/ASO
BIM-FAQS/PCS
BIM-GSS
```

WORDINDEX(string,n)

Purpose

The WORDINDEX function returns starting position of word number *n* in *string*.

Operands

string	String to operate on
n	Word number

Examples

```
WORDINDEX(' one  two three four  ',2)  ==>  7
WORDINDEX(' one  two three four  ',4)  ==>  17
WORDINDEX('one  two three four  ',1)   ==>  1
WORDINDEX('one  two three four  ',5)   ==>  0
```

Sample Program

```
test='BIM-FAQS/ASO BIM-FAQS/PCS BIM-GSS'
say substr(test,wordindex(test,2),wordlength(test,2))
say word(test,2)
end

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
BIM-FAQS/PCS
BIM-FAQS/PCS
```

WORDLENGTH(string,n)

Purpose

The WORDLENGTH function returns the length of word number *n* in *string*.

Operands

string	String to operate on
n	Word number

Examples

```
WORDLENGTH(' one  two three four  ',2)  ==>  3
WORDLENGTH(' one  two three four  ',4)  ==>  5
WORDLENGTH('one  two three four  ',1)   ==>  3
WORDLENGTH('one  two three four  ',5)   ==>  0
```

Sample Program

```
test='BIM-FAQS/ASO BIM-FAQS/PCS BIM-GSS'
say substr(test,wordindex(test,2),wordlength(test,2))
say word(test,2)
end

1...5...10...15...20...25...30...35...40...45...50...55...60..
BIM-FAQS/PCS
BIM-FAQS/PCS
```

WORD POS(string,target<,start)

Purpose

The WORDPOS function returns 0 if *target* not found in *string*. Otherwise, WORDPOS returns word number.

Operands

string	String to search for *target*.
target	Search argument (series of words).
start	Word to begin search with (default=0).

Examples

```
WORDPOS(' one  two three four  ',2)  ==>  3
WORDPOS(' one  two three four  ',4)  ==>  5
WORDPOS('one  two three four  ',1)   ==>  3
WORDPOS('one  two three four  ',5)   ==>  0
```

Sample Program

```
test='BIM-FAQS/ASO BIM-FAQS/PCS BIM-GSS'
say substr(test,wordindex(test,2),wordlength(test,2))
say word(test,2)
end

--------------------------------------------------------------
```

```
1...5...10...15...20...25...30...35...40...45...50...55...60..
BIM-FAQS/PCS
BIM-FAQS/PCS
```

WORDS(string)

Purpose

The WORDS function returns the number of words in *string*. For example, WORDS('one two three   four ') returns 4.

Operand

string	String to operate on

Examples

```
WORDS(' one  two three four')  ==>  4
WORDS('ABC,def 910')           ==>  2
WORDS('')                      ==>  0
```

Sample Program

```
test='FAQS/ASO FAQS/PCS FLEE'
do i = 1 to words(test)
   say word(test,i)
end

--------------------------------------------------------------

1...5...10...15...20...25...30...35...40...45...50...55...60..
FAQS/ASO
FAQS/PCS
FLEE
```

XRANGE(start,end)

Purpose

The XRANGE function returns a string containing all codes from *start* through *end*. If both start and end are omitted, 256 bytes from X'00' through X'FF' are returned. If start is higher than end, the table is wrapped.

Operands

start	Optional starting character (default is '00'x).
end	Optional ending character (default is 'FF'x).

Examples

```
XRANGE('A','F')       ==>  'ABCDEF'
XRANGE('10'x,'15'x)   ==>  '101112131415'x
XRANGE('ff'x,'02'x)   ==>  'FF000102'x
```

'ABCDEFGHIJKLMNOPWRSTUVWXYZ' is not equivalent to XRANGE('A','Z').

X2B(hstring)

Purpose

The X2B function returns the binary string of the hexadecimal string *hstring*. For example, X2B('f1'x) returns '11110001'.

Operand

hstring	String of valid hexadecimal characters.

Examples

```
X2B('C3'x)       ==>  '11000011'
X2B('f1'x)       ==>  '11110001'
X2B('1')         ==>  '11110001'
X2B('8')         ==>  '11111000'
```

Note: This implementation was done prior to Cowlishaw's *The REXX Language*, 2nd edition, and varies from its description.

X2C(hstring)

Purpose

The X2C function returns the character value of the hexadecimal string *hstring*. For example, X2C('F1F9F8F9') returns '1989'.

Operand

hstring — String of valid hexadecimal characters.

Examples

```
X2C('F1f1 c1')       ==>  '11A'
X2C('F')             ==>  '0F'x
X2C('f1f9F9f2')      ==>  '1992'
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a X2C() function on a non-hexadecimal string. The IMOD terminates and the following message appears:

GRX0015I Error in nnnnnnnn, line x Invalid Hexadecimal or Binary string

To ensure that the string is a binary string and thereby avoid a runtime error, use the DATATYPE function.

X2D(hstring,n)

Purpose

The X2D function returns the decimal value of the hexadecimal string *hstring*. For example, X2D('81') returns '129'.

Operands

hstring — String of valid hexadecimal characters.

n — Length of *hstring*. *n* is optional and *hstring* is unsigned if *n* is omitted. If *n* is specified, *hstring* is signed.

Examples

```
X2D('81')      ==>  '129'
X2D('81',2)    ==>  '-127'
X2D('81',4)    ==>  '129'
X2D('F')       ==>  '15'
```

Error Conditions

It is possible for runtime errors to occur with this function. The most common error is trying to execute a X2D() function on a non-hexadecimal string. The IMOD terminates and the following message appears:

GRX0015I Error in nnnnnnnn, line x Invalid Hexadecimal or Binary string

To ensure that the string is a binary string and thereby avoid a runtime error, use the DATATYPE function.

Appendix A
Sample REXX IMODs

This appendix presents some examples of REXX IMODs.

Sample IMODs

$ADDRESS

$ADDRESS is a sample of the ADDRESS CONSOLE function, which allows an IMOD to issue a reply or AR command. BIM-FAQS/ASO is shipped with the AR command "ADDRESS" as a sample in the FAQSASO command file. To use $ADDRESS, type "ADDRESS command" on the system console or in OP mode from a BIM-FAQS/ASO Online interface.

$ARG

$ARG demonstrates how args are passed to IMODs.

$BEEPASO

$BEEPASO is a sample of the IMOD used to set BIM-FAQS/CALL conditions.

$CICSREP

$CICSREP replies to a CICS partition and is called as an external procedure.

$CMSREP

$CMSREP is a working IMOD that performs a reply or command through an SMSG from another machine. $CMSREP echoes the reply back to the user that invoked $CMSREP, or reports on any errors.

$CONSOLE

$CONSOLE provides a sample of the ADDRESS CONSOLE function, which allows an IMOD to issue a reply or AR command. BIM-FAQS/ASO is shipped with the AR command "CONSOLE" as a sample in the FAQSASO command file. To use $CONSOLE, type "CONSOLE command" on the system console or in OP mode from a BIM-FAQS/ASO Online interface.

$CP

$CP issues VM/CP commands and displays information returned on the VSE console. BIM-FAQS/ASO is shipped with the AR command "CP" in the FAQSASO command file. To use $CP, type "CP cpcmd" on the system or in OP mode from a BIM-FAQS/ASO Online interface.

$CPUUSE

$CPUUSE provides information about a partition(s) use of CPU and SIOS over a given interval. This is useful for finding over utilization that may need to be tuned.

$CYCLE

$CYCLE cycles a CICS terminal controlled by VTAM.

$DC

$DC retrieves specified messages from the VSE hardcopy file and routes them back to a VM/CMS user. This IMOD is invoked through an SMSG to the VSE machine to obtain console displays.

$EOJ

$EOJ echoes the time at end-of-job and sets the global variable &job.pid=''.

$GETVIS

$GETVIS displays GETVIS information by specified partition, or all partitions if no partition ID is provided. BIM-FAQS/ASO is shipped with an AR command "$GETVIS" in the FAQSASO command file. To use $GETVIS, type "$GETVIS" on the system or in OP mode from a BIM-FAQS/ASO Online interface, or issue an SMSG from CMS to display the GETVIS information from the target VSE.

$JOB

$JOB sets the global variable &job.pid=jobname and calls the IMOD $jobtime.

$JOBACCT

$JOBACCT issues and displays job accounting info. BIM-FAQS/ASO is shipped with an AR command "$JOBACCT" in the FAQSASO command file. To use this IMOD, type $JOBACCT on the system or in OP mode from a BIM-FAQS/ASO Online interface, or issue an SMSG from CMS to display the JOBACCT information from the target VSE.

$JOBNAME

$JOBNAME displays a job running in a specified partition, or all partitions if no partition ID is provided. BIM-FAQS/ASO is shipped with an AR command "JOBNAME" in the FAQSASO command file. To use $JOBNAME, type JOBNAME on the system or in OP mode from a BIM-FAQS/ASO Online interface, or issue an SMSG from CMS to display the jobs on a target VSE.

$JOBNREP

$JOBNREP allows replies in BIM-FAQS/ASO through jobname.

$JOBTIME

$JOBTIME displays a count of jobs run since the last start of FAQSAO. It is called on // JOB messages occurring on the system console when the shipped action file is used.

$JOBTIME uses global variables, which are saved between IMOD executions. Global variables begin with an ampersand (&) and are not stem variables; global variables are also CPU-specific.

The AOINIT IMOD sets the variables &aojob, &aodate, and &aotime. AOINIT is run each time the FAQSAO task is enabled.

$LOG

$LOG logs a VSE console message on another VSE.

$MESSAGE

$MESSAGE retrieves messages from the VSE hardcopy file and writing them back to the console. $MESSAGE is invoked by the AR command "MESSAGE" set up by default in the command file FAQSASO provided at installation.

$MSG

$MSG retrieves a specified message from the BIM-FAQS/ASO message file (prior to VSE/ESA version 2) or the IBM message file and displays the messages on the console. BIM-FAQS/ASO is shipped with an AR command "$MSG" in the FAQSASO command file. To use this IMOD, type "$MSG mid" on the system console or in OP mode from a BIM-FAQS/ASO Online interface.

$NODESET

$NODESET sets a global variable based on IST105I or 5B05I.

$PA

$PA allows a more powerful use of generics on POWER ALTERs. BIM-FAQS/ASO is shipped with the AR command "PA" in the FAQSASO command file. To use $PA, type "PA que,..." on the system console or in OP mode from a BIM-FAQS/ASO Online interface. $PA parses the POWER ALTER command and runs the POWER queue to check for generic matches. When a match is found, the command is rebuilt with the matching name and the ALTER is performed.

$PA allows the following generics:

- name*
- name++xx
- name==
- name<<

$PHASE

$PHASE displays a job running in a specified partition, or all partitions if no partition ID is provided. BIM-FAQS/ASO is shipped with a AR command "PHASE" in the FAQSASO command file. To use $PHASE, type "PHASE" on the system or in OP mode from a BIM-FAQS/ASO Online interface, or issue an SMSG from CMS to display the jobs on a target VSE.

$POST

$POST posts an event that is waiting on a predecessor condition of PROD=FAQSASO. BIM-FAQS/ASO is shipped with an AR command "POST" in the FAQSASO command file. To use $POST, type "POST evt" on the system console or in OP mode from a BIM-FAQS/ASO Online interface. $POST can also be invoked with an SMSG.

$POWGET

$POWGET is an example of how to access any POWER member and select by page or line number.

$PWRCMD

$PWRCMD issues and retrieves POWER information on a CMS user console via an SMSG or on the system console via XPCC communication to POWER. To use $PWRCMD, type "PWRCMD" on the system or in OP mode from a BIM-FAQS/ASO Online interface, or issue an SMSG from CMS to display the jobs on a target VSE.

$QT

$QT displays the time in an English message on the console. $QT calls the $TIME IMOD to perform the action function. $QT shows a sample of the call function.

$READCON

$READCON reads data from the console and echoes it back. $READCON is provided only as a sample usage of READCONS().

$REPLID

$REPLID uses the REPLYID() function to echo any outstanding replies to the person that issued the command.

$REPLY

$REPLY replies to a partition based upon the partition identifier rather than the reply ID. BIM-FAQS/ASO is shipped with AR commands 'BG' 'F1' ... 'FB' in the FAQSASO command file. To use $REPLY, type "BG xxxxx" on the system console or in OP mode from a BIM-FAQS/ASO Online interface. $REPLY replies to BG or queues a reply for BG. To clear a queued reply, type PRTY REPLY CLEAR pid.

$SCRIPT

$SCRIPT formats help screens for online panels. $SCRIPT is provided as an example of the real usage of some REXX instructions.

$STATUS

$STATUS displays a job running in a specified partition, or all partitions if no partition ID is provided. BIM-FAQS/ASO is shipped with an AR command "STATUS" in the FAQSASO command file. To use $STATUS, type STATUS on the system or in OP mode from a BIM-FAQS/ASO Online interface, or issue an SMSG from CMS to display the jobs on a target VSE.

$SUBMIT

$SUBMIT shows an example of how to submit t a job directly from an IMOD. This could be sued for BIM$RXBA or as a result of BIM-FAQS/ASO GSFAQS message entries.

$TO

$TO replies to a partition based upon partition identifier rather than the actual reply ID. BIM-FAQS/ASO is shipped with an AR command ">" in the FAQSASO command file. To use $TO, type "> machine data" on the system console or in OP mode from a BIM-FAQS/ASO Online interface.

$VTAM

$VTAM issues and displays a VTAM command. BIM-FAQS/ASO is shipped with an AR command "$VTAM" in the FAQSASO command file. To use $VTAM, type $VTAM on the system or in OP mode from a BIM-FAQS/ASO Online interface, or issue an SMSG from CMS to display the VTAM information from the target VSE.

$ADDRESS Sample IMOD

```
* * * * B E G I N  F I L E * * * *
/*********************************************************************/
/* $ADDRESS REXX PROCEDURE: CREATED  7/26/89  BY BOB SMITH          */
/*          updated 02/20/91 support new asoenv call                */
/*                                                                   */
/* This IMOD is designed as a sample of the address console         */
/* function, which allows a IMOD to issue a reply or AR command.    */
/* BIM-FAQS/ASO is shipped with an A/R command "ADDRESS" as a       */
/* sample in the FAQSASO command file.  To use this exec type       */
/* "ADDRESS command" on the system console or in OP mode from a     */
/* BIM-FAQS/ASO Online interface.                                   */
/*********************************************************************/
arg command                          /* cmd is set to the A/R cmd   */
                                     /* that invoked this IMOD and  */
                                     /* command is set to the cmd   */
                                     /* or reply to be issued to AR */
command = strip(command)             /* strip off leading and       */
                                     /* trailing blanks.            */
say 'ADDRESS' '-' date(w) '-' date() '-' time()

address CONSOLE command              /* issue command               */

if rc¬=0 then do                     /* q. is there non-zero RETCODE*/
         if rc=8 then say 'Asynoc busy'
                 else say 'Address Console failed rc='||rc
         end
exit
```

$ARG Sample IMOD

```
/*******************************************************************/
/* $ARG     REXX PROCEDURE: CREATED  2/20/91  BY BOB SMITH       */
/*                                                               */
/* This IMOD is designed to display args and environment data    */
/* passed to an IMOD on the system console.                      */
/*******************************************************************/

arg cmd                            /* cmd is set to the A/R routine */
                                   /* command that drove this IMOD. */
say '*'||cmd||'*'

x=asoenv()                         /*  get environment              */
say '*'||x||'*'
parse var x env lvl node imod pds type data
cmd='';action='';jobname='';phasename='';pid=''
user='';time=''
select
   when type='$CMD' then cmd=data
   when type='SMSG' then user=data
   when type='ONLN' then user=data
   when type='$MSG' then do
        action=substr(data,1,12)
        pid=substr(data,13,2)
        jobname=substr(data,15,8)
        phasename=substr(data,23,8)
        time=substr(data,31,8)
   end
   when type='PCS' then do
        event=substr(data,1,12)
        diskargs=substr(data,13,2)
        group=substr(data,15,8)
        data=substr(data,23,8)
   end
   otherwise nop
end
say 'ENVIRONMENT' env
say 'IMOD       ' imod
say 'Node       ' node
say 'user       ' user
say 'cmd        ' cmd
say 'action     ' action
say 'pid        ' pid
say 'jobname    ' jobname
say 'phasename  ' phasename
say 'time       ' time

exit
```

$BEEPASO Sample IMOD

```
/*****************************************************************/
/* $BEEPER  REXX PROCEDURE: Created  01/09/90 by Dan Shannon     */
/* Function:  Satisfy BIM-FAQS/PCS BEEPER condition              */
/*****************************************************************/
parse upper arg msg
/* --------------------------------------------*/
/*           get shift information            */
/*                                            */
/*      if BIM-FAQS/PCS is present, this      */
/*      section could be replaced by a call to */
/*      KDATE use PCS PROCESSING periods.      */
/* --------------------------------------------*/
d=date(w)
if d¬='Saturday' & d¬='Sunday' then d='Workday'
                               else d='Nonworkday'
day=right(date(o),2)
month=date(M)
select
  when month='January' then do
                            if day='1' then d='Holiday'
                         end
  when month='December' then do
                           if day='25' then d='Holiday'
                         end
  otherwise nop;
end
t=time()
t=substr(t,1,2)||substr(t,4,2)
if t>='0800' & t<'1730'  then shift='Day'
if t>='1730' & t<'2400'  then shift='Night'
if t>='0000' & t<='0100' then shift='Night'
if t>='0100' & t<'0800'  then shift='Home'
/* --------------------------------------------*/
/*        get environment information         */
/* --------------------------------------------*/
x=asoenv()
parse var x env lvl avl imod pds type data
/* --------------------------------------------*/
/*        find out what trigered imod         */
/* --------------------------------------------*/
select
   /* --------------------------------------------*/
   /*        Imod trigered by console message     */
   /* --------------------------------------------*/
   when type='$MSG' then do
        action=substr(data,1,12)
        pid=substr(data,13,2)
        jobname=substr(data,15,8)
        phasename=substr(data,23,8)
        msgid = word(substr(msg,8),1)
        if msgid='1S78I' then do
           msg =strip(strip(substr(msg,38,23),'b'," "),,"'")
           msg = msgid jobname msg
           if d='Workday' then do
              select
                 when shift='Day' then Cond='1'     /* call 1 day    */
                 when shift='Night' then Cond='11'  /* call 1 night  */
                 when shift='Home' then Cond='21'   /* call 2 home   */
                 otherwise exit
              end
           end
           else do
              Cond = '31'                  /* calllist 1 weekend     */
           end
        end
        else do
           if shift¬='Day' then exit       /* do not call if not day */
           if d='Workday' then cond='2'    /* calllist 2 day time    */
                          else cond='32'   /* calllist 2 home        */
                          msg = msgid jobname
         end
   end
   /* --------------------------------------------*/
   /*        Imod trigered by ar command          */
   /* --------------------------------------------*/
   when type='$CMD' then do
       if d='Workday' then do
          select
          when shift='Day' then Cond='3'    /* calllist 3 day time  */
          when shift='Night' then Cond='13' /* calllist 3 week night*/
          when shift='Home' then Cond='23'  /* calllist 3 late night*/
          otherwise exit
          end
       end
       else do
          Cond = '33'                       /* calllist 3 weekend   */
       end
   end
   /* --------------------------------------------*/
   /*        Imod trigered by SMSG                */
   /* --------------------------------------------*/
   when type='SMSG' then do
       user=data
       msg = msg user
       if d='Workday' then do
          select
          when shift='Day' then Cond='3'    /* calllist 3 day time  */
          when shift='Night' then Cond='13' /* calllist 3 week night*/
          when shift='Home' then Cond='23'  /* calllist 3 late night*/
          otherwise exit
          end
       end
       else do
          Cond = '33'                       /* calllist 3 weekend   */
       end
   end
   /* --------------------------------------------*/
   /*        Imod trigered by Event               */
   /* --------------------------------------------*/
   otherwise do
       if d='Workday' then do
          select
          when shift='Day' then Cond='4'    /* calllist 4 day time  */
          when shift='Night' then Cond='14' /* calllist 4 week night*/
          when shift='Home' then Cond='24'  /* calllist 4 late night*/
          otherwise exit
          end
       end
       else do
          Cond = '34'                       /* calllist 4 weekend   */
       end
   end
/* end select */
end
  say 'Beeper condition' cond 'SET by' type '(' msg
  call $beeper 'SET' cond msg
  if result¬='0' then say '$BEEPER Failed RC='||result
exit
```

$CICSREP Sample IMOD

```
/*******************************************************************/
/* $cicsrep REXX PROCEDURE: CREATED  8/21/90  BY BOB SMITH         */
/*                          updated 12/01/92  BY BOB SMITH         */
/*                                            support jobname      */
/* This IMOD is designed to reply to a cics partition and to be    */
/* called as an external procedure.                                */
/*                                                                 */
/* FORMAT 1:   Call $cicsrep pid 'reply'                           */
/*                                                                 */
/* FORMAT 2:   Call $cicsrep jobname 'reply'                       */
/*                                                                 */
/*******************************************************************/
arg pid data                      /* pid is set to the partition to */
                                  /* reply to, and data is the     */
                                  /* reply                         */

x=asoenv()
parse var x . vers .
if vers='ES2' then faqcmd='ASO'
else faqcmd='PRTY'

jobname=''
partitions=pidlist('B')
if length(pid)=2 then do
   do i = 1 to length(partitions) by 2
   if substr(partitions,i,2)=pid then do
      jobname=pid
      leave
      end
   end
end
else jobname=pid

   do i = 1 to length(partitions) by 2
      if jobname=jobname(substr(partitions,i,2)) then do
         pid=substr(partitions,i,2)
         leave
      end
   end

p=phase(pid)                      /*                               */
j=jobname(pid)                    /*                               */
x=replid(pid)                     /*                               */
                                  /*                               */
if rc=0 then do                   /*                               */
   /* ---------------------------------------------- */
   /* ------------ CICS/ICCF partition -------------- */
   /* ---------------------------------------------- */
   if p='DFHSIP' | p='DTSINIT' |p='BIM$UTTS' then do
      if x='' then do
         if p='DFHSIP' then address console 'MSG' pid
                       else address console '/tc'
         do i=1 to 10 until done
            z=wait('1')
            done=replid(pid)¬=' '
         end

         if ¬done then do
            say j 'did not respond to ''MSG' cmd||'''- retry later'
            exit
         end
      end
      repl=strip(substr(replid(pid),4,4))
      address console repl data
```

```
         do i=1 to 10 until done
            z=wait('1')
            done=replid(pid)¬=' '
         end
         if done then do
                     z=wait('1')
                     address console repl
                end
      end
      /* ------------------------------------------------ */
      /* ---------- Not a cics partition ---------------- */
      /* ------------------------------------------------ */
      else do
         if x='' then do
            say 'There are no replies outstanding for' pid
            say 'Reply will be held until required by' pid
            say 'To cancel issue' faqcmd 'REPLY CANCEL'
            address console faqcmd 'REPLY' pid data
         end
         else do
            reply=strip(substr(x,4,4))
            address console repl data
         end
      end
   end
   else do
           say 'Replid failure rc='||d2x(rc)
   end
   exit
```

$CMSREP Sample IMOD

```
/*******************************************************************/
/* $CMSREP  REXX PROCEDURE: CREATED  7/26/89  BY BOB SMITH         */
/*                                                                 */
/* This IMOD is designed as a working IMOD to do a reply or command */
/* via SMSG on from another machine. This exec will echo the       */
/* reply back to the USER that invoked this exec, or report on any  */
/* errors.                                                         */
/*                                                                 */
/* FORMAT:   SMSG machine ASO $CMSREP cmd/reply                    */
/*                                                                 */
/*           machine   is the VSE machine name                     */
/*           ASO       is an identifier for BIM-FAQS/ASO to trigger */
/*                     that an IMOD name is the next blank delimited*/
/*                     parm.                                       */
/*           $CMSREP   The IMOD to executed.                       */
/*           cmd/reply The command or reply to be issued to VSE     */
/*                                                                 */
/* Note:  This may be accomplished via the following format without */
/*        the use of a rexx IMOD                                   */
/*        FORMAT:   SMSG machine OP cmd/reply                      */
/*******************************************************************/
arg command
fuser=strip(substr(word(cpuid(),2),1,8)) /* get vm machine name      */
x=asoenv()                          /* get environment              */
parse var x . . . imod . type user  /* see $args for description    */

if type ¬='SMSG' then do            /* q. was this invoked vis SMSG */
                 say '$CMSREP error - improper environment call:' type
                 exit
             end
```

```
command = strip(command)           /* strip off leading and         */
                                   /* trailing blanks.              */
say  '>' imod user command         /* echo command to console       */
                                   /* ------------------------------*/
                                   /* issue a cp command to display */
                                   /* information to the cms user.  */
                                   /* note: a stem variable z. is   */
                                   /*       used on the CP function, */
                                   /*       this is required since  */
                                   /*       this function is designed*/
                                   /*       to return data back to  */
                                   /*       the caller.             */
                                   /* ------------------------------*/

address CONSOLE command            /* issue the command via asynoc  */

if rc=0 then do                    /* check for any errors          */
        /* echo command or reply to user */
        say   'smsg' user 'ASO $LOG <' fuser command
        z.=cp('smsg' user 'ASO $LOG <' fuser command)
  end
  else do                         /* check for any errors          */
     /* Report error back to user vis cp msg                     */
     if rc¬=8 then z.=cp('smsg' user 'ASO $LOG <' fuser 'Asynoc busy')
             else ,
z.=cp('smsg' user 'ASO $LOG <' fuser 'Address Console failed rc='||rc)

      end                          /* end error check do            */
exit
```

$CONSOLE Sample IMOD

```
/*******************************************************************/
/* $CONSOLE REXX PROCEDURE: CREATED  7/26/89  BY BOB SMITH         */
/*         updated 02/20/91 support new asoenv call                */
/*                                                                 */
/* This IMOD is designed as a sample of the address console        */
/* function, which allows a IMOD to issue a reply or AR command.   */
/* BIM-FAQS/ASO is shipped with a A/R command "CONSOLE" as a sample */
/* in the FAQSASO command file.  To use this exec type             */
/* "CONSOLE command" on the system console or in OP mode from a    */
/* BIM-FAQS/ASO Online interface.                                  */
/*******************************************************************/
arg command                         /* cmd is set to the A/R cmd   */
command = strip(command)            /* strip off leading and       */
                                    /* trailing blanks.            */
x=asoenv()                          /* get environment             */
parse var x . . . . . type user     /* see $args for description   */

   say 'CONSOLE' '-' date(w) '-' date() '-' time()
   if command ¬='' then do
   address CONSOLE command          /* issue the command via asynoc*/

   if rc¬=0 then do                 /* check for any errors        */
         /* Report error back to user                              */
         if rc¬=8 then say 'Asynoc busy'
                  else say 'Address Console failed rc='||rc
         end                        /* end error check do          */
    end
    else say 'No command provided'
exit
```

$CP Sample IMOD

```
/*******************************************************************/
/* $CP      REXX PROCEDURE: CREATED  7/26/89  BY BOB SMITH         */
/*                                                                 */
/* This IMOD is designed to issue and display VM/CP commands       */
/* and information on the VSE console.                             */
/* BIM-FAQS/ASO is shipped with a A/R command "CP" in the FAQSASO  */
/* command file. To use this exec type CP cpcmd on the system      */
/* or in op mode from a BIM-FAQS/ASO online interface.             */
/*                                                                 */
/* CP Command                                                      */
/*                                                                 */
/* CP DISConnect  -  contains support for VM/XA disconnect doc     */
/*                   mode console. IBM does not support under      */
/*                   VM/XA pre VSE/SP4. So this IMOD is provided.  */
/*******************************************************************/
arg cpcmd                         /* cpcmd is the CP command to    */
                                  /* issue.                        */

cpcmd = strip(cpcmd)              /* ensure no leading or trailing */
                                  /* blanks.                       */
say 'VM/CP' '-' date(w) '-' date() '-' time()

if cpcmd='' then do               /* if user did not pass us a CP  */
   say 'No CP command specified'  /* command then report and just  */
   exit                           /* exit the IMOD                 */
   end

z.=''                             /* set the STEM variable to Null */
```

```
                                   /* check to see if user did DISC */
if abbrev('DISCONNECT',cmd,4)=0 then do
    z.=cp(cpcmd)                   /* set the STEM variable to what */
                                   /* the vm/cp command returns     */
                                   /* Note: z.0 is number of STEMS  */
                                   /*       that are set for loop   */

    if rc=0 then do                /* cp function completed         */
                 do i=1 to z.0     /* loop for number of lines      */
                    if z.i ='' then leave  /* sample leave usage    */
                    say strip(z.i) /* echo data to the console      */
                 end               /* end loop                      */
            end                    /* end do                        */
            else do                /* else report error to console  */
                 say 'CP failed rc='||rc
            end                    /* end do                        */
   end
   else do                          /* We have DISC...              */
      z.=cp('Q V CONS')             /* find Virtual console         */
      if word(z.1,1)='CONS' then do /* did we get right data ?      */
         vcons=right(word(z.1,2),3) /* y. then parse 2nd word for 3 */
         z.=cp('RESET' vcons)       /* issue cp reset on VCONS      */
         z.=cp('DISC')              /* issue cp disc                */
         end                        /* end do                       */
   end                              /* end else do                  */
exit
```

$CPUUSE Sample IMOD

```
/*********************************************************************/
/* $CPUUSE  REXX PROCEDURE: CREATED 11/01/01  BY Ken Meyer           */
/*                                                                   */
/* This IMOD is designed to allow you to determine what partition    */
/* or partitions is using a large amount of CPU by issuing multiple  */
/* $JOBACCT commands every x seconds over a period of y seconds and  */
/* then reporting on the results.                                    */
/*                                                                   */
/* VSE                                                               */
/* FORMAT:   $CPUUSE pid interval duration                           */
/*                                                                   */
/*                                                                   */
/*           $CPUUSE   The IMOD to executed.                         */
/*           pid       partition id to check or ALL                  */
/*           interval  number of seconds between requests            */
/*           duration  number of seconds to use as a sample          */
/*                                                                   */
/*********************************************************************/

say '$CPUUSE ' '-' date(w) '-' date() '-' time()
hdr1='Ptn Job      Phase      Intv   CPU     '
hdr1=hdr1||' Start      CPU        SIOS'
hdr2='Id  Name     Name       Secs   100s    '
hdr2=hdr2||' I/Os       /Sec       /Sec'
hdr3='--  ----     ----       ----   ----    '
hdr3=hdr3||' ----       ----       ----'
hflg=1

arg pid intv duratn                  /* get args                     */
pid = substr(strip(pid),1,2)         /* parse partition id           */

pidlist=pid
```

```
if intv=''   | datatype(intv)  <>'NUM' then intv=0
if duratn='' | datatype(duratn)<>'NUM' then duratn=30
if intv<1 then intv=duratn

loopcnt=duratn%intv+1
if loopcnt<2 then loopcnt=2

do cnt=1 to loopcnt
   strt=time('R')
   if pidlist='AL' then pid=pidlist('B')
   j=length(pid)%2
   do i = 0 to j-1
      y=substr(pid,i*2+1,2)
      x=jobacct(y)
      if rc=0 then do
         parse upper var x 1 jnam 10 pnam 19 jtim stim cpus sios
         parse var jtim jhr'.'jmn'.'jse
         parse var stim shr'.'smn'.'sse
         parse var cpus cse'.'cms
         if strip(jhr)='' then iterate
         tcpu=cse||left(cms,2,'0')
         jsec=(jhr*60+jmn)*60+jse
         ssec=(shr*60+smn)*60+sse
         isecs=0
         icpus=0
         icsta=0
         issta=0
         jnam=left(strip(jnam),8)
         pnam=left(strip(pnam),8)
         if jobname.y<>jnam then do
            jobname.y=jnam
            phaname.y=pnam
         end
         else if jsec>jobtime.y then do
            isecs=jsec-jobtime.y
            icpus=tcpu-cpusecs.y
            icsta=(icpus/isecs*1000+5)%10/100
            isios=sios-numsios.y
            issta=(isios/isecs*1000+5)%10/100
         end
         ptn=left(y,3)
         jobtime.y=jsec
         stetime.y=ssec
         cpusecs.y=tcpu
         numsios.y=sios
         if cnt>1 then do
            parse var icsta nm'.'dc
            icsta=right(nm,7,' ')||'.'||left(dc,2,'0')
            parse var issta nm'.'dc
            issta=right(nm,7,' ')||'.'||left(dc,2,'0')
            isecs=right(isecs,6)
            icpus=right(icpus,6)
            isios=right(isios,8)
            if hflg then do
               hflg=0
               say hdr1
               say hdr2
               say hdr3
            end
            say ptn jnam pnam isecs icpus isios icsta issta
         end
      end
end i                /* end user did not specify pid  */
   if cnt<loopcnt then do
      parse value time('E') with wtim'.'msec
      if wtim='' then wtim=0
      if left(msec,1)>='5' then wtim=wtim+1
      wtim=intv-wtim
      z=wait(wtim)
   end
end cnt
say '$CPUUSE Done'

exit
```

$CYCLE Sample IMOD

```
/*********************************************************************/
/* $CYCLE   REXX PROCEDURE: CREATED  8/21/90  BY Dan Curwin         */
/*          updated 02/20/91 support new asoenv call                */
/*          updated 12/01/92 support jobname call                   */
/*                                                                  */
/* This IMOD is designed to cycle a cics terminal controlled by     */
/*   VTAM.                                                          */
/*                                                                  */
/* FORMAT:   CYCLE pid TERMINAL                                     */
/*           CYCLE F2 L080                                          */
/*                                                                  */
/* FORMAT:   CYCLE jobname terminal                                 */
/*           CYCLE PRODCICS L080                                    */
/*********************************************************************/
arg pid data pcst                  /* pid is set to the partition to */
                                   /* data is the terminal to cycle */
term=' '                           /*                               */
node=' '                           /*                               */
pcs=1                              /* set pcs on                    */
if pcst='NOPCS' then pcs=0         /* testing purposes, this IMOD   */
                                   /* validates PCS dynamically      */
call validpid                      /* validate pid specified        */
     if rc¬=0 then exit
/* ------------------------------------------------------------- */
/*       check it termid or nodname: else error                  */
/* ------------------------------------------------------------- */
select
   when length(data) = 4 then term=data
   when length(data) > 0 then node=data
   otherwise do
      x='BIM-FAQS/ASO- You must specify a term-id or Nodename'
      x=x 'for $CYCLE'
        exit
     end
end

/* -------------------------------------------------------------- */
/*  take terminal out of service in cics:                         */
/* -------------------------------------------------------------- */
if term ¬=' ' then do
   say 'Term' term 'is being cycled in partition' pid
   x=pid 'CEMT S TER(' || term ||') OUT'
   call cicscmd                    /*  perform cics command         */
   call getnode                    /*  parse node from cics command */
   if rc¬=0 then do
      say ,
     'BIM-FAQS/ASO- Term-id' term 'undefined/not in session with CICS'
      exit
   end
```

```
end
w=wait('1')                          /* wait 1 second to allow comp. */
/* ------------------------------------------------------------- */
/*  terminal out of service in cics: not inactivate in vtam.        */
/* ------------------------------------------------------------- */
disable='V NET,INACT,ID='||node||',F'
enable='V NET,ACT,ID='||node
say 'Node' node 'is being cycled in partition' pid
call vtamcmd
/* ------------------------------------------------------------- */
/*  If term defined then bring back into service with acquire.       */
/* ------------------------------------------------------------- */
if term¬=' ' then do
   x= pid 'CEMT S TER('||term||') INS ACQ'
   call cicscmd
end
say 'Cycle of' term node 'completed in partition' pid
exit

/* page eject */
/* ------------------------------------------------------------- */
/*                        cicscmd:                                  */
/* ------------------------------------------------------------- */
cicscmd:
cics.=''
if pcs=1 then address cics x      /* try BIM-FAQS/PCS jclrcics       */
                                  /* interface.  This set the stem   */
                                  /* var cics.                       */
if ¬datatype(cics.0,'N') then pcs=0
if rc¬=0 | pcs=0 then do          /* Pcs not enabled or error        */
         pcs=0                    /* set pcs invalid                 */
         call $cicsrep x          /* do it the hard way ....         */
         cics.=''                 /* clear stem variable             */
         cics. = message(pid,'8') /* read the console                */
   end
return

/* page eject */
/* ------------------------------------------------------------- */
/*                          get NODE                                */
/* ------------------------------------------------------------- */
getnode:
node=''
if pcs=1 then do
   do i=1 to cics.0 while node=''
      if word(cics.i,1)='TER('||term||')' then do
           j=i+1
           parse var cics.j . 'NET(' node ')'
           j=i+2
           if node='' then  parse var cics.j . 'NET(' node ')'
         end
   end
end
else do
   do i=1 to cics.0
      if word(cics.i,3)='Ter('||term||')' then
         do
           j=i-1
           parse var cics.j . 'Net(' node ')'
           if node='' then do
                j =i-2
                parse var cics.j . 'Net(' node ')'
              end
           leave
         end
   end
```

```
end
   node=strip(node)
   if node=' ' then rc=16
return
/* page eject */
/* ----------------------------------------------------------------- */
/*  issue vtam command with interface on console commands            */
/* ----------------------------------------------------------------- */
vtamcmd:
/* ----------------------------------------------------------------- */
/* the following command requires the vtam applid for FAQSVSPO       */
/* be defined to vtam and the subtask FAQSVSPO be running in         */
/* the same partition as FAQSASO. If this is not the case,            */
/* this exec will rely on console commands to cycle the node.        */
/* ----------------------------------------------------------------- */
z.=vtam(disable,'4')
if rc=0 then do
   do i=2 to z.0
      say 'CYCLE' z.i
   end
   z.=vtam(enable,'4')
   do i=2 to z.0
      say 'CYCLE' z.i
   end
end
else do
   /* -------------------------------------------------------------- */
   /* the following global variable is set via a message action for  */
   /* ist105i or 5B05I which clears the global variable              */
   /* There are actions defined in the supplied FAQSASO action file  */
   /* which runs the IMOD $NODESET to set the appropriate global     */
   /* variable.                                                      */
   /*                                                                */
   /* These actions may be copied to your file simply by editing     */
   /* the entries for VTAM IST105I and VTAM 5B05I entries in         */
   /* the FAQSASO file and then changing the file name at the top    */
   /* right  ==> FAQSASO  <== to your file name and then pressing    */
   /* PF05                                                           */
   /* -------------------------------------------------------------- */
   address console disable
   &node.node='WAITING'

   do i=1 to 10 until &node.node ¬='WAITING'
      w=wait()
   end
   if &node.node='WAITING' then do
      drop &node.node
      say  'BIM-FAQS/ASO Unable to inactivate node' node
      rc=99
      exit
      end
   else do
      address console enable
      y=wait('5')
   end
end
return
/* ----------------------------------------------------------------- */
/*       check for 2 character pid                                   */
/* ----------------------------------------------------------------- */
validpid:
jobname=''
partitions=pidlist('B')
if length(pid)=2 then do
   do i = 1 to length(partitions) by 2
   if substr(partitions,i,2)=pid then do
```

```
      jobname=pid
      leave
      end
   end
end
else jobname=pid

if jobname='' then do
   say 'BIM-FAQS/ASO - You must specify a Ptn id or a Jobname -' pid
   exit
end

do i = 1 to length(partitions) by 2
   if jobname=jobname(substr(partitions,i,2)) then do
      pid=substr(partitions,i,2)
      leave
   end
end

p=phase(pid)

   if rc¬='0' then do
          say 'BIM-FAQS/ASO - Bad PID ' pid
          return
      end

   if p¬='DFHSIP' & p¬='DTSINIT' then do
        say 'BIM-FAQS/ASO - Wrong partition' pid 'not CICS or ICCF'
        rc=16
        return
      end
return
```

$DC Sample IMOD

```
/*********************************************************************/
/* $DC      REXX PROCEDURE: CREATED  9/27/89  BY BOB SMITH           */
/*          updated 02/20/91 support new asoenv call                 */
/*                                                                   */
/* This IMOD is designed to retrieve a specified messages from       */
/* the VSE hardcopy file and route them back to a VM/CMS user.       */
/* This IMOD is invoked via a SMSG to the VSE machine to obtain      */
/* console displays.                                                 */
/*                                                                   */
/* SMSG machine ASO DC pid cnt scan                                  */
/* ASO machine DC pid cnt scan                                       */
/*                                                                   */
/*      machine is the VSE machine name                              */
/*      AO      is an identifier for BIM-FAQS/ASO to trigger that    */
/*              an IMOD name is the next blank delimited parm.       */
/*      DC      The IMOD to executed.                                */
/*      pid     optional partition id. IE (BG F1 f2,...)             */
/*      cnt     number of messages to return. The default is 20.      */
/*      scan    Scan data to retrieve messages that contain the      */
/*              specified scan data.                                 */
/*                                                                   */
/*      Note: periods (.) may be used as a place marker.             */
/*                                                                   */
/* SMSG DEVVSE ASO DC             (display the last 20 messages      */
/* SMSG DEVVSE ASO DC BG          (display 20 msgs from bg)          */
/* SMSG DEVVSE ASO DC . 30        (display the last 30 msgs)         */
/* SMSG DEVVSE ASO DC F9 5 // JOB (display last 5 jobcards in F9)    */
/*                                                                   */
/*********************************************************************/

arg pid cnt scan

pid = substr(pid,1,2)               /* ensure 2 character pid        */
if pid=' ' then pid=''              /* set pid to null if blanks     */
if pid='.' then pid=''              /* if pid is period then set null*/
if cnt='.' then cnt=''              /* if cnt is period then set null*/
if scan='.' then scan=''            /* if scan is '.' then set null  */

say 'MESSAGE' '-' date(w) '-' date() '-' time()

z.=''                               /* set the STEM variable to Null */
z.=message(pid,cnt,scan)            /* set the STEM variable to with */
                                    /* desired messages.             */
                                    /* Note: z.0 is number of STEMS  */
                                    /*       that are set for loop   */

if rc=0 then do                     /* Message found                 */
             z.0=z.0+1-1
             do i=z.0 to 1 by -1    /* loop backwards for the number */
                                    /* of lines returned             */
                say substr(z.i,1,80)
             end                    /* end loop                      */
        end                         /* end do for message found      */
        else do                     /* else we had a function failure*/
                say 'MESSAGE failed rc='||rc
        end                         /* end do for message failure    */
exit
```

$EOJ Sample IMOD

```
/*******************************************************************/
/* $EOJ     REXX PROCEDURE: CREATED  9/04/90  BY Bob Smith         */
/*                                                                 */
/* This IMOD is designed echo the time at eoj and set the global   */
/* variable &job.pid                                               */
/*                                                                 */
/*   at eoj &job.pid=''                                            */
/*   at job &job.pid='jobname'                                     */
/*                                                                 */
/*******************************************************************/
x=asoenv()                           /*  get environment           */
parse var x env lvl avl imod pds type data
pid=''
   if type='$MSG' then do
        pid=substr(data,13,2)
   end
&job.pid=''                          /* set global variable to null  */
call $time
say result
exit
```

$GETVIS Sample IMOD

```
/*******************************************************************/
/* $GETVIS  REXX PROCEDURE: CREATED  6/24/90  BY BOB SMITH         */
/*          updated 02/20/91 support new asoenv call               */
/*                                                                 */
/* This IMOD is designed to display getvis information by partition  */
/* specified or all partitions if no partition id is provided.     */
/* BIM-FAQS/ASO is shipped with an A/R command "$GETVIS" in the    */
/* FAQSASO command file. To use this exec type $GETVIS on the system */
/* or in op mode from the online interface or issue an SMSG from   */
/* CMS to display the getvis information for the target VSE.       */
/*                                                                 */
/* VSE                                                             */
/* FORMAT:   $GETVIS  pid                                          */
/*                                                                 */
/*          $GETVIS   asynoc command defined in a CONSOLE command  */
/*                    definition file.                             */
/*          pid       Optional partition id, if not specified      */
/*                    all partitions will be displayed.            */
/* CMS                                                             */
/* FORMAT:   SMSG machine ASO $GETVIS pid                          */
/*          ASO  machine GETVIS pid                                */
/*                                                                 */
/*          machine   is the VSE machine name                      */
/*          ASO       is an identifier for BIM-FAQS/ASO to trigger  */
/*                    the IMOD and pass parms                      */
/*                                                                 */
/*          $GETVIS  The IMOD to be executed.                      */
/*          pid       Optional partition id, if not specified      */
/*                    all partitions will be displayed.            */
/*******************************************************************/

arg pid                              /* get pid                    */
pid = substr(strip(pid),1,2)         /* parse partition id         */
if pid='SV' then pid='AR'
x=asoenv()                           /* get environment            */
parse var x . . . . . type user      /* see $args for description  */

say 'GETVIS' '-' date(w) '-' date() '-' time()
say '   Length    Max       Free      Used      Max'
say '   Getvis    Used      Getvis    Getvis    Block'
```

```
if pid¬='' then do                  /* user specified a PID          */
                x=getvis(pid)       /* get getvis for specified pid  */
                if rc=0 then do     /* echo pid and getvis           */
                        y=pid
                        call getdspl y x
                        end
                                    /* else echo error               */
                        else say 'Jobname failed rc='||rc 'pid='||pid
   end                              /* end user specified pid        */

   else do                          /* user did not specify pid      */
      pid=pidlist('B')||'AR'        /* set possible pids           */

             j=length(pid)%2        /* calculate # of Pids in string */
             do i = 0 to j-1        /* loop for number of pids-1     */
               y=substr(pid,i*2+1,2) /* calculate index to pid strng */
               x=getvis(y)          /* get getvis for pid y          */
                if rc=0 then do     /* echo pid and getvis           */
                        call getdspl x
                 end
                                     /* else echo error              */
                        else say 'Getvis failed rc='||rc 'pid='||y
             end                    /* end user did not specify pid  */
   end
exit
/* page eject */                    /* page eject for faqsutil print */
/*********************************************************************/
/*  getdspl: get display information.                                */
/*********************************************************************/
getdspl:
parse var x ctllen maxused free used contig
total=free+used
 total=substr(strip(total,l,'0')||'K',1,9)
 maxused=substr(strip(maxused,l,'0')||'K',1,9)
 free=substr(strip(free,l,'0')||'K',1,9)
 used=substr(strip(used,l,'0')||'K',1,9)
 contig=substr(strip(contig,l,'0')||'K',1,9)
if total='K' then say 'Getvis Area for' y 'Not Initialized'
             else say y total maxused free used contig
return
```

$JOB Sample IMOD

```
/*********************************************************************/
/* $job     REXX PROCEDURE: CREATED  9/04/90  BY Bob Smith           */
/*                                                                   */
/* This IMOD sets the global variable &job.pid and calls the imod    */
/* $jobtime                                                          */
/*                                                                   */
/*   at eoj &job.pid=''                                              */
/*   at job &job.pid='jobname'                                       */
/*                                                                   */
/*********************************************************************/
x=asoenv()                           /*  get environment             */
parse var x env lvl avl imod pds type data
pid=''
   if type='$MSG' then do
        pid=substr(data,13,2)
        job=substr(data,15,8)
   end
&job.pid=job                         /* set global variable to null  */
say &job.pid
call $jobtime
exit
```

$JOBACCT Sample IMOD

```
/*********************************************************************/
/* $JOBACCT REXX PROCEDURE: CREATED 10/16/90  BY BOB SMITH           */
/*          updated 02/20/91 support new asoenv call                 */
/*                                                                   */
/* This IMOD is designed to issue and display job accounting info.   */
/* BIM-FAQS/ASO is shipped with a A/R command "$JOBACCT" in the      */
/* FAQSASO command file. To use this exec type $JOBACCT on the       */
/* system or in op mode from the online interface or issue an SMSG   */
/* from CMS to display the JOBACCT information from the target VSE.  */
/*                                                                   */
/* VSE                                                               */
/* FORMAT:   $JOBACCT  pid                                           */
/*                                                                   */
/* CMS                                                               */
/* FORMAT:   SMSG machine ASO $JOBACCT pid                           */
/*           ASO  machine JOBACCT pid                                */
/*                                                                   */
/*           machine   is the VSE machine name                       */
/*           ASO       is an identifier for BIM-FAQS/ASO to trigger  */
/*                     that an IMOD name is the next blank delimited */
/*                     parm.                                         */
/*           $JOBACCT  The IMOD to executed.                         */
/*           pid       command to display jobacct information        */
/*********************************************************************/
arg pid                              /* get pid                      */
pid = substr(strip(pid),1,2)         /* parse partition id           */
say '$JOBACCT' '-' date(w) '-' date() '-' time()
say '   Job       Phase     Job       Step         Cpu        SIO'
say '   Name      Name      Duration  Duration     Seconds    Count'

if pid¬='' then do                  /* user specified a PID          */
                x=jobacct(pid)      /* get JOBACCT for specified pid */
                if rc=0 then do     /* echo pid and jobacct          */
                        y=pid
                        say y x
                        end
```

```
                                    /* else echo error                */
                        else say 'JOBACCT failed rc='||rc 'pid='||pid
   end                              /* end user specified pid         */

   else do                          /* user did not specify pid       */
      pid=pidlist('B')              /* set possible pids              */
             j=length(pid)%2        /* calculate # of Pids in string */
             do i = 0 to j-1        /* loop for number of pids-1      */
               y=substr(pid,i*2+1,2) /* calculate index to pid string*/
               x=jobacct(y)         /* get JOBACCT for pid y          */
                if rc=0 then do     /* echo pid and jobacct           */
                        say y x
                 end
                                    /* else echo error                */
                        else say 'JOBACCT failed rc='||rc 'pid='||y
             end                    /* end user did not specify pid  */
   end
exit
```

$JOBNAME Sample IMOD

```
/*********************************************************************/
/* $JOBNAME REXX PROCEDURE: CREATED  7/26/89  BY BOB SMITH           */
/*         updated 02/20/91 support new asoenv call                  */
/*                                                                   */
/* This IMOD is designed to display a job running in a partition     */
/* specified or all partitions if no partition id is provided.       */
/* BIM-FAQS/ASO is shipped with a A/R command "JOBNAME" in the       */
/* FAQSASO command file. To use this exec type JOBNAME on the system */
/* or in op mode from a the online interface or issue an SMSG        */
/* from CMS to display the jobs on a target VSE.                     */
/*                                                                   */
/* VSE                                                               */
/* FORMAT:   JOBNAME  pid                                            */
/*                                                                   */
/*           JOBNAME   asynoc command defined in a CONSOLE command   */
/*                     definition file.                              */
/*           pid       Optional partition id, if not specified       */
/*                     all partitions will be displayed.             */
/* CMS                                                               */
/* FORMAT:   SMSG machine ASO $JOBNAME pid                           */
/*           ASO  machine JOBACCT pid                                */
/*                                                                   */
/*           machine   is the VSE machine name                       */
/*           ASO       is an identifier for BIM-FAQS/ASO to trigger  */
/*                     that an IMOD name is the next blank delimited */
/*                     parm.                                         */
/*           $JOBNAME  The IMOD to be executed.                      */
/*           pid       Optional partition id, if not specified       */
/*                     all partitions will be displayed.             */
/*********************************************************************/
arg pid                            /* get pid                        */
pid = substr(strip(pid),1,2)       /* parse partition id             */
say 'JOBNAME' '-' date(w) '-' date() '-' time()

if pid¬='' then do                 /* user specified a PID           */
                x=jobname(pid)     /* get jobname for specified pid  */
                if rc=0 then say pid x  /* echo pid and jobname      */
                                        /* else echo error           */
                        else say 'Jobname failed rc='||rc 'pid='||pid
   end                             /* end user specified pid         */

   else do                         /* user did not specify pid       */
      pid=pidlist('B')             /* set possible pids              */
             j=length(pid)%2       /* calculate # of Pids in string  */
             do i = 0 to j-1       /* loop for number of pids-1      */
               y=substr(pid,i*2+1,2) /* calculate index to pid string*/
               x=jobname(y)        /* get jobname for pid y          */
                if rc=0 then say y x /* echo pid and jobname         */
                                   /* else echo error                */
                        else say 'Jobname failed rc='||rc 'pid='||y
             end                   /* end user did not specify pid   */
   end
exit
```

$JOBNREP Sample IMOD

```
/*****************************************************************/
/* $JOBNREP REXX PROCEDURE:  CREATED 03/28/01  BY Ken Meyer      */
/*                                                               */
/* This IMOD is designed to reply to a partition based upon      */
/* partition jobname rather than the actual reply id.            */
/* This IMOD will reply to the partition with the job name       */
/* provided.  If no message is outstanding, the reply will be    */
/* QUEUED.  To clear a QUEUED reply, type PRTY REPLY CLEAR pid   */
/* on pre-vse/esa 2 systems, or type ASO REPLY CLEAR pid         */
/*****************************************************************/

arg jnam rep                        /* cmd is set to the A/R cmd     */
                                    /* that invoked this IMOD and    */
                                    /* rep is set to the reply       */
x=asoenv()
parse var x . vers . imod . type pid
if vers='ES2' then faqcmd='ASO'
else faqcmd='PRTY'
if type¬='$CMD' then do
   say imod 'only supported via an AR command'
   exit
end
jnam=translate(jnam)
jct=0
pid=pidlist('B')                    /* set possible pids             */
j=length(pid)%2                     /* calculate # of Pids in string */
do i = 0 to j-1                     /* loop for number of pids-1     */
   y=substr(pid,i*2+1,2)            /* calculate index to pid string */
   x=jobname(y)                     /* get jobname for pid y         */
   if x=jnam then do
      jct=jct+1
      jtab.jct=y                    /* put partition id in table     */
   end
end i                               /* end user did not specify pid  */
if jct=0 then do
   say 'There are no partitions with the jobname:' jnam
   exit
end
pid=jtab.1
if jct>1 then do
   say 'There are multiple partitions with the jobname:' jnam
   say 'select one from the following list:'
   plist=''
   pwork=''
   do i=1 to jct
      plist=plist||jtab.i||' '
      pwork=pwork||jtab.i||' '
      if i//20=0 then do
         say plist
         plist=''
      end
   end i
   if plist<>'' then say plist
   pid=''
   do while pid=''
      x=readcons('Enter partition id to select')
      if x=' ' then do
         say 'The reply to' jnam 'was ignored'
         exit
      end
      y=wordpos(translate(x),pwork)
      if y>0 then pid=jtab.y
   end
end
rlen=3
if vers='ES2' then rlen=4
x=replid(pid)
if rc=0 then do
   x=strip(x)
   if length(x)>7 then do
      say 'There are multiple reply ids for the partition:' pid
      say 'select one from the following list:'
      rlist=''
      rwork=''
      do i = 1 to words(x)
         rlist=rlist||strip(substr(word(x,i),4))||' '
         rwork=rwork||strip(substr(word(x,i),4))||' '
         if i//10=0 then do
            say rlist
            rlist=''
         end
      end
      if rlist<>'' then say rlist
      x=''
      do while x=''
         x=readcons('Enter reply id to select')
         if x=' ' then do
            say 'The reply to' pid 'was ignored'
            exit
         end
         x=right(x,rlen,'0')
         if wordpos(x,rwork)=0 then x=''
      end
   end
   else x=strip(substr(x,4))
   if x=' ' then do
      say 'No outstanding replies for' pid 'reply is queued'
      ADDRESS CONSOLE faqcmd 'REPLY' pid rep
   end
   else ADDRESS CONSOLE x rep
end
else do
   say 'REPLID FAILURE RC='d2x(rc)
end

exit                                 /* exit IMOD                     */
```

$JOBTIME Sample IMOD

```
/*****************************************************************/
/* $JOBTIME REXX PROCEDURE: CREATED  2/10/90  BY BOB SMITH       */
/*                                                               */
/* This IMOD is designed to display a count of jobs run since    */
/* the last start of FAQSAO.   It is called on // JOB messages   */
/* occurring on the system console when the shipped action file   */
/* is used.  This exec uses global variables which are saved     */
/* between IMOD executions.  They are CPU specific.  Global       */
/* variables begin with an & and are not a stem variable.        */
/* The IMOD AOINIT sets the variables &aojob, &aodate and        */
/* &aotime.  This exec is run each time the FAQSAO task is       */
/* enabled.                                                      */
/*****************************************************************/

   if datatype(&aojob)¬='NUM' then &aojob=0
      /* precaution for null variable or bad variable. */
      &aojob=&aojob+1               /* increment the job count       */

   say 'This is the start of job number' &aojob 'since' &aodate &aotime
   exit
   return
```

$LOG Sample IMOD

```
/*****************************************************************/
/* $LOG   REXX PROCEDURE: Created  6/13/90  By Bob Smith         */
/*          updated 02/20/91 support new asoenv call             */
/*          updated 03/17/93 add address ao say console          */
/*        Log a message from one VSE to another                  */
/*****************************************************************/
  arg msg
  address AO 'SAY CONSOLE'
  x=length(msg)-10
  if x>70 then do
             y=substr(msg,x,8)
             if substr(y,3,1)=':' & substr(y,6,1)=':' then do
                     msg=strip(substr(msg,1,x-1))
                 end
         end
  say msg
exit
```

$MESSAGE Sample IMOD

```
/*********************************************************************/
/* $MESSAGE REXX PROCEDURE: CREATED  9/27/89  BY BOB SMITH           */
/*          updated 02/20/91 support new asoenv call                 */
/*                                                                   */
/* This IMOD is designed as a sample for retrieving messages         */
/* the VSE hardcopy file and writing them back to the console.       */
/* This IMOD is invoked by the A/R command MESSAGE set up by         */
/* default in the command file FAQSASO provided at install.          */
/*                                                                   */
/*      MESSAGE pid cnt scan                                         */
/*                                                                   */
/*      pid     optional partition id. IE (BG F1 f2,...)             */
/*      cnt     number of messages to return. The default is 20.     */
/*      scan    Scan data to retrieve messages that contain the      */
/*              specified scan data.                                 */
/*                                                                   */
/*      Note: periods (.) may be used as a place marker.             */
/*                                                                   */
/*      MESSAGE              (display the last 20 messages           */
/*      MESSAGE BG           (display 20 msgs from bg)               */
/*      MESSAGE . 30         (display the last 30 msgs)              */
/*      MESSAGE F9 5 // JOB  (display last 5 jobcards in F9)         */
/*                                                                   */
/*********************************************************************/

arg pid cnt scan                    /* cmd is set to DC, user is the */
                                    /* cms user that did smsg to run */
                                    /* this IMOD. PID is the         */
                                    /* partition to display or null  */
                                    /* or period for a filler.  Cnt  */
                                    /* is the number of messages to  */
                                    /* return. Scan is scan data     */

pid = strip(substr(pid,1,2))        /* ensure 2 character pid or null*/
if pid='.' then pid=''              /* if pid is period then set null*/
if cnt='.' then cnt=''              /* if cnt is period then set null*/
if scan='.' then scan=''            /* if scan is '.' then set null  */

say 'MESSAGE' '-' date(w) '-' date() '-' time()

z.=''                               /* set the STEM variable to Null */

z.=message(pid,cnt,scan)            /* set the STEM variable to with */
                                    /* desired messages.             */
                                    /* Note: z.0 is number of STEMS  */
                                    /*       that are set for loop   */

if rc=0 then do                     /* Message found                 */


             do i=1 to z.0          /* loop for each message         */
                                    /* ------------------------------*/
                                    /* loop for each message assigned*/
                                    /* in the stem variable. Messages*/
                                    /* are placed in the stem in LIFO*/
                                    /* order, since they are built   */
                                    /* as the hardcopy file is read  */
                                    /* backwards from the current    */
                                    /* point.  There is a sample IMOD*/
                                    /* $DC which displays the        */
                                    /* in the standard visual output */
                                    /* format.                       */
                                    /* ------------------------------*/
                say substr(i,1,2) substr(z.i,1,68)
             end                    /* end loop                      */
        end                         /* end message                   */
        else do                     /* else report error             */
             say 'MESSAGE failed rc='||rc
        end                         /* end do                        */
exit
```

$MSG Sample IMOD

```
/*********************************************************************/
/* $MSG  REXX PROCEDURE: CREATED 12/11/89  BY BOB SMITH              */
/*                                                                    */
/* This IMOD is designed to retrieve a specified message from         */
/* the BIM-FAQS/ASO message VSAM file (pre=vse/esa 2), or the IBM     */
/* message file and display it on the console.  BIM-FAQS/ASO is       */
/* shipped with a A/R command "$MSG" in the FAQSASO command file.     */
/* To use this exec type "$MSG mid" on the system console or in OP    */
/* mode from the online interface.                                    */
/**********************************************************************/

arg mid .                            /* cmd is set to the A/R cmd     */
                                     /* that invoked this IMOD and    */
                                     /* mid is set to the message to  */
                                     /* echoed to the system console  */
say 'MSG' '-' date(w) '-' date() '-' time()

z.=''                                /* set the STEM variable to Null */
z.=msg(mid)                          /* set the STEM variable to MSG  */
                                     /* Note: z.0 is number of STEMS  */
                                     /*       that are set for loop   */

if rc=0 then do                      /* Message found                 */
             do i=1 to z.0           /* loop for number of lines      */
                say strip(z.i,t)     /* Echo to the console but strip */
                                     /* trailing blanks first         */
             end                     /* end loop                      */
        end                          /* end do                        */

        else do                      /* Error on msg()                */
             say 'MSG failed rc='||rc  /* echo return code on console*/
        end                          /* end do                        */
exit                                 /* exit IMOD                     */
```

$NODESET Sample IMOD

```
/*****************************************************************/
/* $NODESET REXX PROCEDURE: Created  8/27/90  By Bob Smith       */
/*                                            and Andy Jezerski   */
/*        set a global variable based upon IST105I               */
/*        and 5B05I                                              */
/*****************************************************************/
  arg msg
  /* MSG=F3 003 IST105I D72L802 NODE ...                          */
  /* MSG=F3 003 5b05i d772L802 NODE ..                            */
  mnum=substr(msg,8,5)               /* get message number trigger */
  if mnum='5B05I' then node=substr(msg,14,8)
                  else node=substr(msg,16,8)
  parse var node node .              /* get rid of any extra chars */
                                     /* if node name lt 8 chars.   */
  node=strip(node)                   /* ensure leading and trailing*/
                                     /* blanks gone                */
  &node.node=''                      /* set node stem to null      */
exit
```

$PA Sample IMOD

```
/*****************************************************************/
/*****************************************************************/
/* $PA      REXX PROCEDURE: CREATED  1/18/90  BY BOB SMITH       */
/*                                                               */
/* This IMOD is designed to allow a more powerful use of         */
/* generics on POWER ALTERS. BIM-FAQS/ASO is shipped with an A/R */
/* command PA in the FAQSASO command file. To use the exec type  */
/* "PA que,..." on the system console or in OP mode from the     */
/* online interface. This exec will parse the POWER alter        */
/* command and run the POWER queue to check for generic matches  */
/* When a match is found the command is rebuilt with the         */
/* matching name and the alter is performed.                     */
/*                                                               */
/* This IMOD allows generics of:                                 */
/*                                                               */
/* name*                                                         */
/* name++xx                                                      */
/* name==                                                        */
/* name<<                                                        */
/*                                                               */
/*****************************************************************/
  arg data
  data=translate(data,' ',',')   /* parse data into words */
  cjob=''
  error=''
  pque=''
  pjob=''
  pclass=''
  ppri=''
  pfno=''
  fnode=''
  tnode=''
  fuser=''
  tuser=''
  pdisp=''
  psysid=''
  afno=''
  acopy=''
  adest=''
  adisp=''
  anode=''
  auser=''
  aclass=''
  asysid=''
  aremote=''
  acmpact=''

  parse var data pque data
  /* parse power parms */
  do i=1 to words(data)

     pwrparm=word(data,i)
     select;
     /* power search parms   */
     when 'CPRI= '  =left(pwrparm,5) then ppri=right(pwrparm,1)
     when 'CFNO='   =left(pwrparm,5) then pfno=substr(pwrparm,6)
     when 'TNODE='  =left(pwrparm,6) then tnode=substr(pwrparm,7)
     when 'FNODE='  =left(pwrparm,6) then fnode=substr(pwrparm,7)
     when 'TUSER='  =left(pwrparm,6) then tuser=substr(pwrparm,7)
     when 'FUSER='  =left(pwrparm,6) then fuser=substr(pwrparm,7)
     when 'CDISP='  =left(pwrparm,6) then pdisp=right(pwrparm,1)
     when 'CCLASS=' =left(pwrparm,7) then pclass=right(pwrparm,1)
     when 'CSYSID=' =left(pwrparm,7) then psysid=right(pwrparm,1)
     /* power action parms   */
     when 'FNO='    = left(pwrparm,4) then afno=pwrparm
     when 'PRI='    = left(pwrparm,4) then apri=pwrparm
     when 'COPY='   = left(pwrparm,5) then acopy=pwrparm
     when 'DEST='   = left(pwrparm,5) then adest=pwrparm
     when 'DISP='   = left(pwrparm,5) then adisp=pwrparm
     when 'NODE='   = left(pwrparm,5) then anode=pwrparm
     when 'USER='   = left(pwrparm,5) then auser=pwrparm
     when 'CLASS='  = left(pwrparm,6) then aclass=pwrparm
     when 'SYSID='  = left(pwrparm,6) then asysid=pwrparm
     when 'REMOTE=' = left(pwrparm,7) then aremote=pwrparm
     when 'CMPACT=' = left(pwrparm,7) then acmpact=pwrparm
     when 'CRDATE=' = left(pwrparm,7) then acmpact=pwrparm
     when 'CRDATE<' = left(pwrparm,7) then acmpact=pwrparm
     when 'CRDATE>' = left(pwrparm,7) then acmpact=pwrparm
     otherwise do
            if length(pwrparm)=1 then class=pwrparm
               else do
               if pjob='' then do
                               class=''
                               generic=' '
                               pjob=pwrparm
                               if substr(pjob,1,1)='*' then do
                                  cjob=substr(pjob,2,8)
                                  generic='*'
                               end
                               else do
                                  pjob='*'||pjob
                                  cjob=substr(pjob,2,8)
                                  generic='*'
                               end
                               cjob=strip(translate(cjob))
                               pjob=strip(translate(pjob,' ','*+<='))
                               pjob='*'||word(pjob,1)
                               if pjob='*' then pjob=''
                          end
                          else do
                               if length(pwrparm)=1 then class=pwrparm
                                           else error='INVALID JOBNAME'
                          end
               end
     end
     end
end
     if error='' then do
        /* issue power command */
        if pjob='' then pjob='ALL'
        z.=power(pque,pjob,class)
        /* ---------------------------- */
        /*  loop for each match on data   */
        /* ---------------------------- */
        do i=1 to z.0
           ojob=substr(z.i,1,8)
           ojnum=strip(substr(z.i,10,5))
           match=1
           if generic='*' then k=length(cjob)
                          else k=length(ojob)
              do j=1 to k
                 if cjob¬='' then leave j
                 cjobj=substr(cjob,j,1)
                 ojobj=substr(ojob,j,1)
                 if substr(ojob,j,1)¬=cjobj then do
                    select;
                    when ojobj=cjobj then nop
                    when cjobj='+' then nop
                    when cjobj='<' then do
                            if ojobj<='A' | ojobj>='Z' then do
                            match=0
                            leave j
                            end
                         end
                    when cjobj='=' then do
                            if ojobj<='0' | ojobj>='9' then do
                            match=0
                            leave j
                            end
                         end
                    otherwise do
                        match=0
                        leave j
                        end
                    end
                 end
              end
           /* ------------------------------ */
           /*  perform checks on each match  */
           /* ------------------------------ */
           if match=1 then do
              /* ------------------------------ */
              /*  perform more checks           */
              /* ------------------------------ */
              /* pclass
                 pfno  ppri pdisp
                 psysid
                 tnode fnode
                 tuser fuser*/
              /* ------------------------------ */
              /*  built power alter command     */
              /* ------------------------------ */
              say 'Alter' substr(z.i,1,60)
              pcmd='A '||pque||','||strip(ojob)||','||ojnum
              if afno¬=''    then pcmd=pcmd||','||afno
              if acopy¬=''   then pcmd=pcmd||','||acopy
              if adest¬=''   then pcmd=pcmd||','||adest
              if adisp¬=''   then pcmd=pcmd||','||adisp
              if anode¬=''   then pcmd=pcmd||','||anode
              if auser¬=''   then pcmd=pcmd||','||auser
              if aclass¬=''  then pcmd=pcmd||','||aclass
              if asysid¬=''  then pcmd=pcmd||','||asysid
              if aremote¬='' then pcmd=pcmd||','||aremote
              if acmpact¬='' then pcmd=pcmd||','||acmpact
              address console  pcmd
              end
              else do
                nop;
              end
        end
     end
     else do
        say error
     end

say 'done:'

exit
```

$PHASE Sample IMOD

```
/**********************************************************************/
/* $PHASE REXX PROCEDURE: CREATED  7/26/89  BY BOB SMITH             */
/*          updated 02/20/91 support new asoenv call                */
/*                                                                  */
/* This IMOD is designed to display a phase name running in a       */
/* partition specified or all partitions if no partition id is      */
/* provided.  BIM-FAQS/ASO is shipped with a A/R command "PHASE"    */
/* in the FAQSASO command file. To use this exec type PHASE on the  */
/* system or in op mode from the online interface or issue an       */
/* SMSG from CMS to display the jobs on a target VSE.               */
/*                                                                  */
/* VSE                                                              */
/* FORMAT:   PHASE     pid                                          */
/*                                                                  */
/*           PHASE     asynoc command defined in a CONSOLE command  */
/*                     definition file.                             */
/*           pid       Optional partition id, if not specified      */
/*                     all partitions will be displayed.            */
/* CMS                                                              */
/* FORMAT:   SMSG machine ASO $PHASE pid                            */
/*                                                                  */
/*           machine   is the VSE machine name                      */
/*           AO        is an identifier for BIM-FAQS/ASO to trigger  */
/*                     that an IMOD name is the next blank delimited */
/*                     parm.                                        */
/*           $PHASE    The IMOD to executed.                        */
/*           pid       Optional partition id, if not specified      */
/*                     all partitions will be displayed.            */
/*******************************************************************/
arg pid                            /* get pid                      */
pid = substr(strip(pid),1,2)       /* parse partition id           */

say 'PHASE' '-' date(w) '-' date() '-' time()

if pid¬='' then do                 /* user specified a PID          */
                x=phase(pid)       /* get phase for specified pid   */
                if rc=0 then say pid x  /* echo pid and phase       */
                                    /* else echo error              */
                        else say 'Phase failed rc='||rc 'pid='||pid
   end                             /* end user specified pid        */

   else do                         /* user did not specify pid      */
      pid=pidlist('B')             /* set possible pids             */
      j=length(pid)%2              /* calculate # of Pids in string */
      do i = 0 to j-1              /* loop for number of pids-1     */
         y=substr(pid,i*2+1,2)     /* calculate index to pid string */
         x=phase(y)                /* get phase for pid y           */
         if rc=0 then say y x      /* echo pid and phase            */
                                   /* else echo error               */
                 else say 'Phase failed rc='||rc 'pid='||y
     end                           /* end user did not specify pid  */
   end
exit
```

$POST Sample IMOD

```
/*********************************************************************/
/* $POST REXX PROCEDURE: CREATED 08/10/89  BY BOB SMITH              */
/*          updated 02/20/91 support new asoenv call                 */
/*                                                                   */
/* This IMOD is designed post an event that is waiting on a          */
/* predecessor condition of PROD=FAQSASO.                            */
/* BIM-FAQS/ASO is shipped with a A/R command "POST"                 */
/* in the FAQSASO command file.  To use this exec type               */
/* "POST evt" on the system console or in OP mode from the           */
/* Online interface. This IMOD may also be invoked via SMSG          */
/*********************************************************************/

arg event .                         /* cmd is set to the A/R cmd     */
                                    /* that invoked this IMOD and    */
                                    /* event is set to the event to  */
                                    /* be posted                     */
event = strip(substr(event,1,8))    /* set event to max of 8         */
say 'POST' '-' date(w) '-' date() '-' time()
x=post(event)                       /* post event                    */
                                    /* report event status           */
if rc=0 then say 'Event' event 'posted'
        else say 'POST' event 'failed rc='||rc
exit
```

$POWGET Sample IMOD

```
/*********************************************************************/
/* $powget REXX PROCEDURE: CREATED  6/06/01  BY KEN MEYER            */
/*                                                                   */
/* This IMOD is an example of use of the $powget POWER member read   */
/* function available with BIM-FAQS (ASO & PCS).  This routine will  */
/* return up to 200 lines of a POWER RDR/LST/PUN member and display  */
/* them on the console.  This function can start on any page or line */
/* within the requested member.                                      */
/*                                                                   */
/* VSE                                                               */
/* FORMAT:   $powget   que jobname jobnumber suffix sysid '(' rest   */
/*                                                                   */
/*          que       name of the POWER queue to be searched.        */
/*          jobname   name of the POWER rdr, lst, pun, etc           */
/*                    member.                                        */
/*          number    number of member that corresponds to the       */
/*                    jobname described above.                       */
/*          suffix    segment number within the requested job        */
/*                    number.                                        */
/*          sysid     system id this member belongs to.              */
/*          rest      search information consisting of:              */
/*                    P=nnnnnn  - where nnnnnn is the page or line   */
/*                    L=nnnnnn    number to search for               */
/*                    C=nnnnnn    number of lines to get             */
/*                                                                   */
/*********************************************************************/
parse arg qtype data
parse upper var data jobname number suffix sysid '(' rest

jobname=left(jobname,8,' ')
if datatype(number)<>'NUM' then number=0
if datatype(suffix)<>'NUM' then suffix=0
if datatype(sysid) <>'NUM' then sysid=0
if suffix=0 then suffix=""
if sysid=0 then sysid=""

flg1='00'x
flg2='00'x
tnum='00000000'x
cnt=200

do while rest<>''
   parse var rest p'='dat rest
   if rest='' then parse var dat dat','rest
   select
      when p='C' then do
         if datatype(dat)='NUM' then cnt=dat
      end
      when pos(p,'PL')<>0 then do
         if datatype(dat)='NUM' then do
            tnum=d2c(dat,4)
            if p='P' then flg1=bitor(flg1,'40'x)
            else flg1=bitor(flg1,'80'x)
         end
      end
      otherwise nop
   end
end

flg1=bitor(flg1,'20'x)

x=substr(qtype,1,1)||left(jobname,8)
if number¬='' then x=x||d2c(number,2)
else x=x||'0000'x
if suffix¬='' then x=x||d2c(suffix,1)
else x=x||'00'x
if sysid¬=''  then x=x||d2c(sysid,1)
else x=x||'00'x
x=x||'00'x             /* class */
x=x||tnum||flg1||flg2

z.=powget('O',x)
if rc<>0 then do
   say 'Member '||jobname number suffix||' not found in' qtype
   say 'GAO607 OPEN FAILED RC='||d2x(rc)
   exit 999
end

eod=0
do while eod=0 & cnt>0
   call process
   if eod=0 & cnt>0 then do
      z.=powget('R')
      if rc<>0 then do
         if rc=9 then z.0=0
         else do
            say 'GAO607 POWGET ROUTINE FAILED RC='||d2x(rc)
            exit 999
         end
      end
   end
end

z.=powget('C')

exit 0
```

```
process:
 nlins=z.0
 do i=1 to nlins while cnt>0
    len=c2d(left(z.i,2,'00'x))
    flg=substr(z.i,5,1)
    rln=len-5-1                      /* remove hdr, ctl              */
    if bitand(flg,'80'x)='80'x then do
       rln=rln-4
       pln=substr(z.i,len-3,4)
    end
    else pln='00000000'x
    pln=right(c2d(pln),10)
    if rln<0 then exit 9999
    ctl=substr(z.i,6,1)
    rec=substr(z.i,7,rln)
    if ctl='8B'x | ctl='FE'x then pln='P='||right(pln,8)
    if bitand(flg,'01'x)='01'x then do            /* ASA conversion?  */
       say c2x(ctl)
       ctl='01'x
    end
    say c2x(ctl) rec '>' pln
    if right(pln,2)<>' 0' then cnt=cnt-1
 end
 if nlins<200 then eod=1
return
```

$PWRCMD Sample IMOD

```
/*********************************************************************/
/* $PWRCMD  REXX PROCEDURE: CREATED  7/26/89  BY BOB SMITH           */
/*          updated 02/20/91 support new asoenv call                 */
/*                                                                   */
/* This IMOD is designed to issue and retrieve power information on  */
/* a cms user console via SMSG or on the system console via          */
/* xpcc communication to POWER.                                       */
/* To use this exec type PWRCMD on the system or in op mode from     */
/* a BIM-FAQS/ASO online interface or issue an SMSG from CMS to      */
/* display the jobs on a target VSE.                                 */
/*                                                                   */
/* VSE                                                               */
/* FORMAT:   PWRCMD cmd                                              */
/*                                                                   */
/*           cmd       valid power cmd which may be issued via       */
/*                     xpcc                                          */
/* CMS                                                               */
/* FORMAT:   SMSG machine ASO $PWRCMD cmd                            */
/*                                                                   */
/*           machine   is the VSE machine name                       */
/*           AO        is an identifier for BIM-FAQS/ASO to trigger  */
/*                     that an IMOD name is the next blank delimited */
/*                     parm.                                         */
/*           $POWER    The IMOD to be executed.                      */
/*           cmd       valid power cmd which may be issued via       */
/*                     xpcc                                          */
/*********************************************************************/
arg pcmd                            /* pcmd is pwrcmd to issue       */

z.=''
say 'PWRCMD' que '-' date(w) '-' date() '-' time()
   z.=pwrcmd(pcmd)
   if rc=0 then do
      do i=1 to z.0
         say strip(z.i)
      end
   end
exit
```

$QT Sample IMOD

```
/*****************************************************************/
/* $QT      REXX PROCEDURE: CREATED  7/26/89  BY BOB SMITH       */
/*                                                               */
/* This IMOD is designed to display the time in English on the   */
/* console. It calls the rexx IMOD $TIME to perform the action   */
/* function.  This shows a sample of the call function.          */
/*****************************************************************/

call $TIME                      /* call the IMOD $TIME to get    */
                                /* the TOD in English.           */
                                /* this function return a string */
                                /* in result.                    */

say result                      /* display on console.           */
exit
```

$READCON Sample IMOD

```
/*****************************************************************/
/* READCONS REXX PROCEDURE: CREATED  8/10/89  BY BOB SMITH       */
/* READ from console                                             */
/*****************************************************************/
say 'READCONS' '-' date(w) '-' date() '-' time()
x=readcons('Enter some data, it will be echoed to the console')
say 'Data entered was -' x
exit
```

$REPLID Sample IMOD

```
/*****************************************************************/
/* REPLYID  REXX PROCEDURE: CREATED  7/26/89  BY BOB SMITH       */
/*          updated 02/20/91 support new asoenv call             */
/*          Display job names and reply ids for a partition or   */
/*          all partitions if id is not specified.               */
/*                                                               */
/*          if cmd=SMSG then this was called via smsg from cms   */
/*          user, so we will reply to him via msg/msgnoh         */
/*****************************************************************/
arg pid                            /* get pid                        */
pid = strip(substr(strip(pid),1,2)) /* parse partition id            */

     say 'REPLY ID' '-' date(w) '-' date() '-' time()
     x=replid(pid)
     if rc=0 then do
        if x=' ' then do
           say 'There are no outstanding replies'
        end
        else do
         pid=pidlist('B')||'AR'     /* set possible pids              */
               j=length(pid)%2      /* calculate # of Pids in string */
               do i = 0 to j-1      /* loop for number of pids-1     */

                 p=substr(pid,i*2+1,2)
                 x=replid(p)
                 name=jobname(p)
                 if x¬=' ' then do
                     say name x
                    x.=message(p,'2')
                    do j = 1 to x.0
                       say x.j
                    end
                 end
               end
        end
     end
     else do
          say 'REPLID failed rc='||rc
     end
exit
```

$REPLY Sample IMOD

```
/*****************************************************************/
```

```
/* $REPLY REXX PROCEDURE: CREATED 03/18/90  BY BOB SMITH         */
/* Modified 2/11/91 - check for multiple replies                 */
/* Modified 2/18/91 - support new args with asoenv call          */
/* Modified 7/01/92 - display better message on multiple replies */
/*                                                               */
/* This IMOD is designed to reply to a partition based upon      */
/* partition identifier rather than the actual reply id.         */
/* BIM-FAQS/ASO is shipped with A/R commands 'BG' 'F1' ... 'FB'  */
/* in the FAQSASO command file.  To use this exec type           */
/* "BG xxxxx" on the system console or in OP mode from the       */
/* online interface. It will reply to BG or QUEUE a reply        */
/* for BG.  To clear a QUEUED reply, type PRTY REPLY CLEAR pid   */
/* on pre-vse/esa 2 systems, or type ASO REPLY CLEAR pid         */
/*****************************************************************/

arg rep                              /* cmd is set to the A/R cmd     */
                                     /* that invoked this IMOD and    */
                                     /* rep is set to the reply       */
x=asoenv()
parse var x . vers . imod . type pid
if vers='ES2' then faqcmd='ASO'
else faqcmd='PRTY'
if type¬='$CMD' then do
      say imod 'only supported via an AR command'
      exit
   end
x=replid(pid)
if rc=0 then do
        x=strip(x)
        if length(x)>7 then do
            y='There are multiple replies outstanding for'
            y=y pid 'which task do you want'
            z=''
            do i = 1 to words(x)
                z=z strip(substr(word(x,i),4,4))
            end
            x=readcons(y z)
            if x=' ' then do
               say 'The reply to' pid 'was ignored'
               exit
            end
            end
        x=strip(substr(x,4,4))
        if x=' ' then do
    say ,
    'There are no replies outstanding for' pid||', the reply is queued'
           ADDRESS CONSOLE faqcmd 'REPLY' pid rep
           end
           else do
                address console x rep
           end
   end
   else do
        say 'REPLID FAILURE RC='d2x(rc)
   end

exit                                 /* exit IMOD                     */
```

$SCRIPT Sample IMOD

```
/*********************************************************************/
/* $script   REXX PROCEDURE: CREATED 05/27/92  BY BOB SMITH         */
/*                                                                   */
/* This IMOD is designed format help screens for online panels       */
/*                                                                   */
/* the following global variables must be set to run this processor. */
/*                                                                   */
/*         &si=1;&soffset='';&sfo='';&slen=''                        */
/*         &d.=''                                                    */
/*                                                                   */
/*    x.1 ='.ce.header'                                              */
/*    x.2 ='Specify which\job|name (from COMREG)'                    */
/*    x.3 ='which must be executing when'                            */
/*    x.4 ='the message occurs.'                                     */
/*    x.5 ='.sk'                                                     */
/*    x.6 ='The following special characters'                        */
/*    x.7 ='are supported to allow for generics:'                    */
/*    x.8 ='.fo no'                                                  */
/*    x.9 ='.sk'                                                     */
/*    x.10='       + (plus sign)      Matches any character'         */
/*    x.11='       = (equal sign)     Matches any numeric character' */
/*    x.12='.fo yes'                                                 */
.tr ¢ :
/*    x.13='¢ul'                                                     */
/*    x.14='¢li.list element 1'                                      */
/*    x.15='¢li.list element 2'                                      */
/*    x.16='¢eul'                                                    */
.tr ¢ ¢
/*    x.17=''                                                        */
/*    /* ------------------------------ */                           */
/*    /*       Call script processor    */                           */
/*    /* ------------------------------ */                           */
/*    &si=1;&soffset='';&sfo='';&slen=''                             */
/*                                                                   */
/*    call $script '.init 30'                                        */
/*                                                                   */
/*    do k = 1 to 100 while x.k¬=''                                  */
/*       call $script x.k                                            */
/*    end                                                            */
/*                                                                   */
/*    call $script '.end'                                            */
/*                                                                   */
/*    )BODY SCROLL                                                   */
/*                       Field Oriented Help                         */
/*    |                                                              */
/*    )TEXT &d.                                                      */
/*    )LL $PF3=Return\                                               */
/*                                                                   */
/*********************************************************************/
parse arg data
select
  when word(data,1)='.init' then do
       call init
       end
  when data='.fo no' then do
       i=&si+1
       &sfo='n'
       end
  when data='.fo yes' then do
       i=&si+1
       &sfo='y'
       end
.tr ¢ :
  when data='¢ul' then do
       i=&si+1
       end
when substr(data,1,4)='¢li.' then do
       data =substr(data,5,length(data)-4)
       data='@@*@'||data
       i=&si
       if &d.i¬='' then i=i+1
       &d.i=' '
       i=i+1
       call listbeg
       end
  when substr(data,1,4)='.ce.' then do
       data =substr(data,5,length(data)-4)
       i=&si
       if &d.i¬='' then i=i+1
       i=i+1
       call scenter
       end
  when data='¢eul' then do
       i=&si
       if &d.i¬='' then i=i+1
       &d.i=' '
       i=i+1
       data=''
       call listend
       end
.tr ¢ ¢
  when word(data,1)='.sk' then do
       i=&si
       if &d.i¬='' then i=i+1
       &d.i=' '
       i=i+1
       end
  when word(data,1)='.end' then do
       i=&si+1
       &d.i=''
       end
  otherwise do
       i=&si
       call reform
       end
end
&si=i
exit
init:
i=1
&slen=word(data,2)
&soffset=''
&sfo='y'
&d.=''
a='n'
data=''
exit

reform:
if &sfo='n' then do
   &d.i=data
   i=i+1
   return
   end
if data='' then return

if length(&d.i)+length(data) < &slen then do
   if &d.i='' then &d.i=data
              else &d.i=&d.i data
   data=''
   return
end
do j=1 to words(data)
   if length(&d.i)+wordlength(data,j) < &slen then do
      if &d.i='' then &d.i=word(data,j)
```

```
                 else &d.i=&d.i word(data,j)
   end
else do
         d=&d.i
         select
            when substr(d,1,4)='    ' then do
                 d='@@@@'||substr(d,5,length(d)-4)
                 d=reverse(d)
                 d=justify(d,&slen)
                 d=reverse(d)
                 d='    '||substr(d,5,length(d)-4)
                 &d.i=d
             end
             when substr(d,1,3)='@@*' then do
                  d='@@@'||substr(d,4,length(d)-3)
                  d=reverse(d)
                  d=justify(d,&slen)
                  d=reverse(d)
                  d='  *'||substr(d,4,length(d)-3)
                  &d.i=d
              end
            otherwise do
                 d=reverse(d)
                 d=justify(d,&slen)
                 &d.i=reverse(d)
            end
         end
      i=i+1
      &d.i=&soffset||word(data,j)
   end
end
data=''
return

/* ---------------------------- */
/*          center:             */
/* ---------------------------- */
scenter:
   l=length(&soffset)
   if &d.i¬='' then i=i+1
   &d.i=&soffset||center(data,&slen-l,' ')
   i=i+1
return
/* ---------------------------- */
/*          listbeg:            */
/* ---------------------------- */
listbeg:
&soffset='    '
   &d.i=data
return
/* ---------------------------- */
/*          listend:            */
/* ---------------------------- */
listend:
&soffset=''
if data¬='' then do
   if substr(data,1,3)='@@*' then do
         data='  *'||substr(data,4,length(data)-3)
   end
   &d.i=data
   data=''
end
   if substr(&d.i,1,3)='@@*' then do
         &d.i='  *'||substr(&d.i,4,length(&d.i)-3)
   end
   i=i+1
```

```
return
```

$STATUS Sample IMOD

```
/*********************************************************************/
/* $STATUS REXX PROCEDURE: CREATED  7/26/89  BY BOB SMITH            */
/*          updated 02/20/91 support new asoenv call                 */
/*                                                                   */
/* This IMOD is designed to display the status of a job running in a */
/* partition specified or all partitions if no partition id is       */
/* specified.  BIM-FAQS/ASO is shipped with an A/R command "STATUS"  */
/* in the FAQSASO command file.  To use this exec type "STATUS" on   */
/* the system console or in op mode from the online interface or     */
/* issue an SMSG from CMS to display the status of jobs on a target  */
/* VSE.                                                              */
/*                                                                   */
/* VSE                                                               */
/* FORMAT:   STATUS    asynoc command defined in a CONSOLE command   */
/*                     definition file.                              */
/*           pid       Optional partition id, if not specified       */
/*                     all partitions will be displayed.             */
/* CMS                                                               */
/* FORMAT:   SMSG machine ASO $STATUS                                */
/*                                                                   */
/*           machine   is the VSE machine name                       */
/*           AO        is an identifier for BIM-FAQS/ASO to trigger  */
/*                     that an IMOD name is the next blank delimited */
/*                     parm.                                         */
/*           $STATUS   The IMOD to executed.                         */
/*********************************************************************/

      say 'STATUS' '-' date(w) '-' date() '-' time()
      pid=pidlist('B')||'AR'        /* set possible pids             */
      j=length(pid)%2               /* calculate # of Pids in string */
      do i = 0 to j-1               /* loop for number of pids-1     */
         say substr(pid,i*2+1,2) status(substr(pid,i*2+1,2))
      end
exit
```

$SUBMIT Sample IMOD

```
/*****************************************************************/
/* $SUBMIT  REXX PROCEDURE: CREATED 06/25/01  BY Ken Meyer       */
/*                                                               */
/* This IMOD shows an example of how to submit a JOB to POWER    */
/* using BIM-GSS REXX.                                           */
/*****************************************************************/

x.1 = '* $$ JOB JNM=TEST,CLASS=0,DISP=D'
x.2 = '* $$ LST CLASS=A'
x.3 = '// PAUSE  TEST SUBMIT'
x.4 = '/*'
x.5 = '/&'
x.6 = '* $$ EOJ'
x.7 = '   '
x.0=7
if userfunc()<>'BIM$SFPA' then if userfunc("BIM$SFPA")='' then do
  say "LOAD FAILED FOR BIM$SFPA"
  exit rc
end
z=pds('OPEN','SUBMIT')
if rc<>0 then
  say 'OPEN Failed RC='||d2x(rc)
do i = 1 to x.0
   z=pds('SUBA',x.i)
end i
z=pds('CLOSE','SUBMIT')
if rc<>0 then do
   say 'CLOSE FAILED RC='d2x(rc)
end
exit rc
```

$TO Sample IMOD

```
/*****************************************************************/
/* $TO    REXX PROCEDURE: CREATED 03/18/90  BY BOB SMITH         */
/*          updated 02/20/91 support new asoenv call             */
/*                                                               */
/* This IMOD is designed to send replies or commands to another  */
/* machine under vm via SMSG.  BIM-FAQS/ASO is shipped with an   */
/* A/R command '>' in the FAQSASO command file.  To use this     */
/* exec type "> machine data" on the system console or in OP     */
/* mode from the online interface.                               */
/*****************************************************************/
arg tuser imod cmd               /* cmd is set to the A/R cmd     */
                                 /* that invoked this IMOD and    */
                                 /* rep is set to the reply       */
x=asoenv()                       /* get environment               */
parse var x . . . . . type .     /* see $args for description     */
if type¬='$CMD' Then do
      say '$TO imod only supported vis AR command'
      exit
   end

if imod='REPLY' then imod='$CMSREP'

if substr(imod,1,1)¬='$' then do
                             cmd=imod cmd
                             imod='$CMSREP'
                        end
z.=cp('SMSG' tuser 'ASO' imod cmd) /* all smsg vse target machine   */
exit                              /* exit IMOD                      */
```

$VTAM Sample IMOD

```
/*********************************************************************/
/* $VTAM    REXX PROCEDURE: CREATED 10/16/90  BY BOB SMITH           */
/*          updated 02/20/91 support new asoenv call                 */
/*                                                                   */
/* This IMOD is designed to issue and display a VTAM command         */
/* BIM-FAQS/ASO is shipped with a A/R command "$VTAM" in the FAQSASO */
/* command file. To use this exec type $VTAM on the system           */
/* or in op mode from the online interface or issue an SMSG          */
/* from CMS to display the VTAM information from the target VSE.     */
/*                                                                   */
/* VSE                                                               */
/* FORMAT:   $VTAM cmd                                               */
/*                                                                   */
/* CMS                                                               */
/* FORMAT:   SMSG machine ASO $VTAM cmd                              */
/*           ASO  machine JOBACCT pid                                */
/*                                                                   */
/*           machine   is the VSE machine name                       */
/*           ASO       is an identifier for BIM-FAQS/ASO to trigger */
/*                     that an IMOD name is the next blank delimited */
/*                     parm.                                         */
/*           $VTAM     The IMOD to executed.                         */
/*           cmd       command to display VTAM information           */
/*********************************************************************/

arg vtamcmd                        /* get vtamcmd                   */
vtamcmd = strip(translate(vtamcmd))
x=asoenv()                         /* get environment               */
parse var x . . . . . type user    /* see $args for description     */
```

```
say '$VTAM' '-' date(w) '-' date() '-' time()

         z.=vtam(vtamcmd,'5')       /* issue vtam command and wait 5 */
           if rc=0 then do
                do i=1 to z.0       /* loop for number of returned msg*/
                   z.i=strip(z.i,T) /* strip trailing blanks         */
                   say z.i          /* echo to console               */
                end
           end
           else do
                say 'VTAM error rc=' rc
           end
exit
```

$WAIT Sample IMOD

```
/*****************************************************************/
/* WAIT     REXX PROCEDURE: CREATED  7/26/89  BY BOB SMITH       */
/*          updated 02/20/91 support new asoenv call             */
/*          This IMOD will issue a wait for n seconds where n    */
/*          is the passed parameter,                              */
/*****************************************************************/
arg wtime .
wtime = strip(wtime)
say 'wait' '-' date(w) '-' date() '-' time()
z=wait(wtime)
say 'wait rc='||rc
exit
```

$WAKEUP Sample IMOD

```
/*****************************************************************/
/* $wakeup  REXX PROCEDURE: CREATED  7/20/89  BY BOB SMITH       */
/*          this IMOD will issue wakeup message every minute,    */
/*          quarter hour, half hour, or hour based on parameter  */
/*****************************************************************/
arg type .
hour=0
quarter=0
half=0
do forever
c8=time()
m=substr(c8,4,2)+0
s=substr(c8,7,2)+0
m=59-m                              /* calculate number of min left  */
s=59-s                              /* wake up on the minute         */
select
   when type='MIN' then do
                   m=0
                   minute=minute+1
                   count=minute
                   end
   when type='HALF' then do
                   if m>29 then m=m-30
                   half=half+1
                   count=half
                   end
   when type='QUARTER' then do
                   if m>14 then m=m-15
                   if m>14 then m=m-15
                   if m>14 then m=m-15
                   quarter=quarter+1
                   count=quarter
                   end
otherwise do
                   hour=hour+1
                   count=hour
                   type='HOUR'
          end
end
s=s+(60*m)
say 'wait' s%60 'minutes and' s-((s%60)*60) 'seconds'
s=value(s)
x=wait(s)
call $TIME                         /* call the IMOD $TIME to get    */
                                   /* the TOD in English.           */
                                   /* this function return a string */
                                   /* in result.                    */
say 'Wakeup:' type 'count='||count result
end

   exit
```

Glossary
Basic Terms

Assignment
Variables can be assigned data by the use of ARG, PARSE, PULL, and ADDRESS environments. You can also use an equal sign (=) to assign a symbol to the value of an expression.

Binary string
A binary string is a string of 0's and 1's, grouped in four characters that can be delimited by one or more blanks. The first string of characters is assumed to have a length of 4, and is right justified with zeros added to align the string to four characters. The string must be delimited on the left with a single quote (') or double quote ("), and on the right with a 'B, 'b, "B, or "b.

Comment
'A comment is delimited with a /* and a */. Comments are free-form, appearing anywhere and spanning any number of lines. Comments are useful for documenting IMODs and for commenting out sections of code.

Expression
An expression is an instance of one or more strings, symbols, operators, or functions. Expressions are evaluated left to right with respect to parentheses and operator precedence.

Hexadecimal string
A hexadecimal string is a string made up of the characters 0-9, A-F, a-f, grouped in pairs delimited by one or more blanks. The first character need not be paired, and a 0 is added on the left to pair this character. The string must be delimited on the left with a single quote (') or double quote ("), and on the right with an 'X, 'x, "X, or "x.

String
A string is a group of characters delimited by single quotes (' ') or double quotes (" "). You can delimit the string with double quotes if you include a single quote in the string, or vice versa. When this is not possible, place two single or two double quotes together to denote a single character. The null string is used many times in REXX and is shown as ''.

Symbol
A symbol is a variable, constant, or keyword made up of the following characters: A-Z, a-z, 0-9, period (.), exclamation mark (!), underscore (_), at sign (@), pound sign (#), question mark (?), and dollar sign ($). Symbols are translated to uppercase before use. For example, the symbols BOB, Bob, and bob are identical.

Template
A template is a list of symbols delimited by blanks and/or patterns. For example: cmd '(' arg1 arg2

Variables
BIM's implementation of REXX contains four types of variables:

- Simple variables
- Stem variables
- Global variables
- Global stem variables

Simple variables
Simple variables are symbols whose values can be changed during the execution of a REXX IMOD. Simple variables can be up to 50 characters long and can be assigned values up to 4096 bytes. These variables are local to the currently executing REXX IMOD and cannot be referenced by other IMODs or procedures. Called procedures can, however, access these simple variables via the EXPOSE command on the procedure definition.

Stem variables
Stem variables are composed of a stem symbol and a period (.), denoting a family of stems that can be cleared, initialized, set, or dropped. You can assign or reference any member of a stem family by appending a symbol to the family name. This symbol can be a constant or a variable. A variable is useful for accessing all members of a family when the members are named by numbers and the 0 member contains the number of members in the family.

Global variables
Global variables are symbols whose values can be changed during the execution of a REXX IMOD. Global

variables can be up to 8 characters long and can be assigned values up to 105 bytes, including the variable name. Global variables are prefixed with an ampersand (&). Because of the nature of REXX, global variables cannot be concatenated through abuttal; they must be concatenated with a blank or ||.

Global variables can be used only for REXX IMODs executed via the FAQSAO task. These IMODs include those triggered using SMSGs, console messages, console commands, online commands, BIM-FAQS/PCS commands, and GEM (Global Event Manager). Global variables are kept until you assign them to null or drop them. They are kept on disk and in storage, and will survive an IPL.

Because of the nature of the FAQSAO task multi-threading IMODs, you can be assured of the state of a global variable until you perform a function that has an implied wait--such as CP(), WAIT(), POWER(), PWRCMD, MESSAGE(), or MSG(). At these implied waits, other IMODS are run, and the values of your global variables are subject to change.

Global stem variables

Global stem variables function like normal stem variables but according to the rules of global variables. The only difference between global variables and global stem variables is their life expectancy: Since global stem variables are not written to disk, they do not survive end-of-job.

Index

$

$ADDRESS IMOD, sample A-1, A-7

$ARG IMOD, sample A-1, A-8

$BEEPASO IMOD, sample A-1, A-9

$CICSREP IMOD, sample A-1, A-11

$CMSREP IMOD, sample A-1, A-12

$CONSOLE IMOD, sample A-2, A-14

$CP IMOD, sample A-2, A-14

$CPUUSE IMOD, sample A-2, A-15

$CYCLE IMOD, sample A-2, A-17

$DC IMOD, sample A-2, A-21

$EOJ IMOD, sample A-2, A-22

$GETVIS IMOD, sample A-22

$GWTVIS IMOD, sample A-2

$JOB IMOD, sample A-3, A-24

$JOBACCT IMOD, sample A-3, A-24

$JOBNAME IMOD, sample A-3, A-26

$JOBNREP IMOD, sample A-3, A-27

$JOBTIME IMOD, sample A-3, A-29

$LOG IMOD, sample A-4, A-29

$MESSAGE IMOD, sample A-4, A-29

$MSG IMOD, sample A-4, A-31

$NODESET IMOD, sample A-4, A-31

$PA IMOD, sample A-4, A-32

$PHASE IMOD, sample A-5, A-35

$POST IMOD, sample A-5, A-36

$POWGET IMOD, sample A-5, A-36

$PWRCMD IMOD, sample A-5, A-39

$QT IMOD, sample A-5, A-39

$READCON IMOD, sample A-5, A-40

$REPLID IMOD, sample A-6, A-40

$REPLY IMOD, sample A-6, A-41

$SCRIPT IMOD, sample A-6, A-42

$STATUS IMOD, sample A-6, A-45

$SUBMIT IMOD, sample A-6, A-46

$TO IMOD, sample A-6, A-47

$VTAM IMOD, sample A-7, A-47

$WAIT IMOD, sample A-48

$WAKEUP IMOD, sample A-48

=

= (REXX editor line command) 1-21

A

ABBREV 5-2

ABS 5-3

Adding ASO files
    REXX IMOD 1-9

ADDRESS AO 4-1

ADDRESS CARD 4-3

ADDRESS CICS 4-4

ADDRESS CONSOLE 4-5

ADDRESS DISK 4-5, 4-6

ADDRESS EVENT 4-8

ADDRESS EXPLORE 4-9

ADDRESS OUTPUT 4-13

ADDRESS PDATE 4-14

ADDRESS PDS 4-15

ADDRESS POWER 4-16

ADDRESS PROGRAM 4-17

ADDRESS SCHEDULE 4-18

ADDRESS SYS 4-19

Advanced command-line commands 1-21

ARG template 3-2

B

B2C(binary string 5-10

B2X(binary string 5-11

Basic command-line commands 1-15

BITAND 5-7

BITOR 5-8

BITXOR 5-9

C

C2D 2-17

CALL function <expression<,expression>...> 3-3

CENTER(string, length<,pad>) 5-12

CENTRE(string, length<,pad>) 5-13

CLUSTER(parms) 5-14

CP(cmd,ASIS) 5-16

CPUID( ) 5-18

D

DATE 2-17

Deleting ASO files
  REXX IMOD 1-9

Differences btw. BIM and IBM REXX
  C2D 2-17
  DATE 2-17
  DUMPSTG 2-17
  Floating point numbers 2-17
  INTERPRET 2-15
  Labels 2-15
  MAX 2-16
  MIN 2-16
  Signal 2-16
  Symbols 2-16
  TRACEs 2-16
  UPPER 2-17

Directory lists
  REXX IMOD 1-7

DO 3-3

DROP 3-5

DUMPSTG 2-17

E

Editing a REXX IMOD 1-12

Editing ASO files
  REXX IMOD 1-10

Editor screens
  REXX IMOD 1-10

ELSE 3-8

Enabling REXX procedure support 1-3

Enabling REXX support 1-3

Entering commands on the command line 1-14

EXIT <expression> 3-6

F

FAQSAO task
  main task 1-4

Files
  REXX IMOD 1-7

Filing an IMOD 1-29

Floating point numbers 2-17

Function help
  accessing 2-6

Function lengths 2-15

Functions

ABBREV 5-2
ABS 5-3
ADDRESS 5-4
ADDRESS AO 4-1
ADDRESS CARD 4-3
ADDRESS CICS 4-4
ADDRESS CONSOLE 4-5
ADDRESS DISK 4-5, 4-6
ADDRESS EVENT 4-8
ADDRESS EXPLORE 4-9
ADDRESS OUTPUT 4-13
ADDRESS PDATE 4-14
ADDRESS PDS 4-15
ADDRESS POWER 4-16
ADDRESS PROGRAM 4-17
ADDRESS SCHEDULE 4-18
ADDRESS SYS 4-19
ARG 5-5
ASOENV 5-6
B2C(binary string) 5-10
B2X(binary string) 5-11
BITAND 5-7
BITOR 5-8
BITXOR 5-9
C2D(string<,n>) 5-18
C2X(string) 5-19
CENTER(string, length<,pad>) 5-12
CENTRE(string, length<,pad>) 5-13
CLUSTER(parms) 5-14
COMPARE(string1,string2,pad) 5-15
COPIES(string,n) 5-16
CP(cmd,ASIS) 5-16
CPUID( ) 5-18
D2C(number<,n>) 5-24
D2X(number<,n>) 5-25
DATATYPE(string<,type>) 5-19
DATE(<option<,date<,'B'>>>) 5-21
DELSTR(string,n<length>) 5-22, 5-23
DIGITS( ) 5-23
ERRORTEXT(n) 5-25
FIND(string,tgt) 5-26
FORM( ) 5-27
FORMAT(number,<integer>,<decimal>) 5-27
FUZZ( ) 5-29
GETVIS(pid) 5-29
INDEX(haystack,needle,start) 5-31
INSERT(new,target,<n>,<length>,<pad>) 5-32
JOBACCT('CPU') 5-34
JOBACCT('PAG') 5-34
JOBACCT(pid) 5-33
JOBNAME(pid) 5-35
JUSTIFY(string,length,pad) 5-36
LASTPOS(needle,haystack,start) 5-37
LEFT(string,length,pad) 5-38
LENGTH(string) 5-39
LIBR(function,parms) 5-40
LINESIZE( ) 5-40
LISTCAT(parms) 5-41
MAX 5-41
MESSAGE 5-42
MIN 5-43
MSG 5-44
OVERLAY 5-45
PHASE 5-46
PIDLIST 5-47
POS 5-48
POST 5-49
POWER 5-49
PWRCMD 5-51
QUEUED 5-52
RANDOM 5-52
READCONS 5-54
REPLID 5-55
REVERSE 5-56
REXX 2-1
RIGHT 5-56
SESSION 5-57
SIGN 5-58
SOURCELINE 5-58
SPACE 5-59
STATUS 5-60
STRIP 5-61
SUBSTR 5-62
SUBWORD 5-63
SYMBOL 5-63
TIME 5-64
TRANSLATE 5-65
TRUNC 5-65
user 2-2
USERID 5-66
VALUE 5-67
VERIFY 5-68
VSAM(function, parms) 5-69
VSSPACE(parms) 5-69
VTAM 5-70
WAIT 5-71
WORD 5-72
WORDINDEX 5-73
WORDLENGTH 5-73
WORDPOS 5-74
WORDS 5-75
X2B 5-76
X2C 5-77
X2D 5-77
XRANGE 5-76

G

GSFAQS commands
  DISABLE AO 1-4

I

IF 3-6

Initialization and configuration
  IMOD initialization 1-5

Instructions
  DO 3-3
  DROP 3-5
  ELSE 3-8
  EXIT <expression> 3-6
  IF 3-6
  ITERATE <symbol> 3-7
  LEAVE <symbol> 3-8
  NOP 3-9
  NUMERIC 3-9
  OTHERWISE 3-9
  PARSE 3-10
  PROCEDURE <expose <(> variable list <)> > 3-11
  PULL 3-11
  PUSH 3-12
  QUEUE 3-12
  RETURN 3-12
  REXX 2-1
  SAY 3-13
  SELECT 3-14
  SIGNAL 3-14
  THEN 3-7
  TRACE All/Commands/Error/Fail/Intermediate/Labels/Normal/Off/Results 3-15
  UPPER 3-15
  WHEN 3-16

INTERPRET 2-15

ITERATE <symbol> 3-7

L

Label lengths 2-15

Labels 2-15

LEAVE <symbol> 3-8

Length
  function 2-15
  label 2-15
  symbol 2-15

LIBR(function,parms) 5-40

LISTCAT(parms) 5-41

Listing ASO files
  REXX IMOD 1-7

Loop control variables 2-15

M

MAX 2-16, 5-41

MESSAGE 5-42

MIN 2-16, 5-43

Modifying ASO files
  REXX IMOD 1-9

MSG 5-44

N

NOP 3-9

NUMERIC 3-9

O

Online help
  accessing 2-4
  example 2-3
  REXX language 2-3

OTHERWISE 3-9

OVERLAY 5-45

P

PARSE 3-10

Performance hints 2-18

PF-key functions
    using 1-26

PHASE 5-46

PIDLIST 5-47

Prefix-area commands 1-13

Procedures
    REXX 1-3

PULL 3-11

PUSH 3-12

Q

QUEUE 3-12

R

READCONS 5-54

REPLID 5-55

RESULT 3-2

RETURN 3-12

REVERSE 5-56

REXX
    definition 1-1
    how to edit 1-12
    SDL/SVA Requirements 1-2
    use of 1-1

REXX compiler 1-29
    error processing 1-30

REXX editor
    basic line commands 1-15
    command-line commands 1-14
    PF-key functions 1-26

REXX editor line commands
    = 1-21
    Add 1-15
    Backward 1-15
    Bottom 1-16
    Case 1-16
    Change 1-17
    DELete 1-17
    Down 1-18
    DUPlicate 1-21
    FFile 1-22
    File 1-19
    GET 1-22
    Input 1-23
    Next 1-18
    Overlay Column 1-23
    PDS 1-24
    Quit 1-24
    Recall 1-21
    SAVE 1-19
    Search 1-19
    SSAVE 1-25
    TAB 1-25
    TOP 1-16
    Up 1-15

REXX IMOD
    directory list 1-7
    Editor screen 1-12
    filing 1-29
    format 1-11
    saving 1-29

REXX IMOD file
    adding 1-9
    deleting 1-9
    editing 1-12
    editor 1-10
    executing 1-9
    listing 1-7
    modifying 1-9

REXX language 2-1
    functions in general 2-1, 2-2
    instructions 2-1
    online help 2-3

REXX support
    enabling 1-3

RIGHT 5-56

S

SAY 3-13

SDL/SVA Requirements 1-2

SELECT 3-14

Signal 2-16

SIGNAL 3-14

SOURCELINE 5-58

SPACE 5-59

STATUS 5-60

STRIP 5-61

SUBSTR 5-62

SUBWORD 5-63

SYMBOL 5-63

Symbol lengths 2-15

Symbols 2-16

T

Terminating BIM-FAQS/ASO 1-4

THEN 3-7

TIME 5-64

TRACE All/Commands/Error/Fail/Intermediate/Labels/Normal/Off/Results 3-15

TRACEs 2-16

TRANSLATE 5-65

TRUNC 5-65

U

UPPER 2-17, 3-15

USERID 5-66

V

VALUE 5-67

Variable 2-11

VERIFY 5-68

VSAM(function, parms) 5-69

VSSPACE(parms) 5-69

VTAM 5-70

W

WAIT 5-71

WHEN 3-16

WORD 5-72

WORDINDEX 5-73

WORDLENGTH 5-73

WORDPOS 5-74

WORDS 5-75

X

X2B 5-76

X2C 5-77

X2D 5-77

XRANGE 5-76