

**Chapter
1**

**ADVANCED
TOPICS**

*Get on the
Fast Track!*



TM

**SYS-ED/
COMPUTER
EDUCATION
TECHNIQUES, INC.**

Objectives

You will learn:

- C Passing input data on strings.
- C Handling error conditions.
- C Debugging commands.
- C Determining data set characteristics.
- C Verifying the existence of a dataset.
- C Getting system information.
- C Pushing and popping a stack on a queue.

1 Parsing Data

The operation of parsing involves taking data and breaking them into fields. The data is a string, an argument or data from a stack.

PULL var1 var2 ...

Retrieves information from the terminal or stack and changes the data to upper case.

PARSE PULL var1 var2 ...

Retrieves information from the terminal or stack and does not change the data

ARG var1 var2 ...

This commands takes information passed as arguments from command line, functions or subroutines.

All the contents are converted to upper case.

PARSE ARG var1 var2 ...

This commands takes information passed as arguments from command line, functions or subroutines.

All the contents are not converted to upper case.

PARSE VAR string var1 var2 ...

This command takes a specified variable and breaks it into multiple variables.

PARSE UPPER VAR charvar var1 var2 ...

This command takes a specified variable and breaks it into multiple variables. All the contents are converted to upper case.

PARSE VALUE string WITH var1 var2 ...

This command breaks a string into multiple parts.

2 Parsing Separators

BLANK	Each variable gets a word from the data. Each word is delimited by a space.
STRING	A string can be specified as the delimiter. A period can be specified as a placeholder. Its associated data is not stored anyplace.
VARIABLE	The separator can be placed in a variable.
ABSOLUTE COLUMN POSITION	The column position can be specified to indicate where the string is to be broken.
RELATIVE COLUMN POSITION	A signed integer separates the data according to the relative column position.

3 PARSE VAR

VAR1 = 'TODAY IS A WORKDAY'

Example 1:

```
PARSE VAR VAR1 W1 W2 W3 W4

      W1 = TODAY
      W2 = IS
      W3 = A
      W4 = WORKDAY
```

Example 2:

```
PARSE VAR VAR1 W1 4 W2 10 W3

      W1 = TOD
      W2 = AY IS
      W3 = A WORKDAY
```

Example 3:

```
PARSE VAR VAR1 W1 6 W2 1 W3

      W1 = TODAY
      W2 = IS A WORKDAY
      W3 = TODAY IS A WORKDAY
```

Example 4:

```
PARSE VAR VAR1 1 W1 5 6 W2 11 W3

      W1 = TODA
      W2 = IS A
      W3 = WORKDAY
```

Example 5:

```
PARSE VAR VAR1 W1 +5 W2 +6 W3

      W1 = TODAY
      W2 = IS A
      W3 = WORKDAY
```

4 SOURCELINE Function

```
SOURCELINE (line-number)
```

The SOURCELINE function allows the program to extract specific lines from the source code and display them.

SOURCELINE () returns the number of lines in the program.

This function is often used to provide a simple HELP facility.

Example:

```
DO I=1
  IF SUBSTR(SOURCELINE(I),1,2)='*/' then leave
  SAY SOURCELINE(I)
END
```

5 MSG Function

Both error and information messages are normally printed when lost commands.

To prevent the messages from being displayed as an REXX program runs use the MSG function.

```
status = MSG('OFF')
```

MSG() returns the status (OFF,ON) of the messages.

6 Debugging Information

TRACE Statement

TRACE I (Intermediate)	TRACE	?I
TRACE R (Results)	TRACE	?R
TRACE N (Normal)	TRACE	?N
TRACE L (Labels)	TRACE	?L
TRACE OFF (TURN IT OFF)		

TRACE OUTPUT

The output of the trace includes:

```
>>>>V>>>> Variable Value
>>>>L>>>> Literal Value
>>>>O>>>> Operation Result
>>>>F>>>> Result Of a Function
```

TRACE ?

Will stop at every statement and display output as above. Pressing the enter key will trace the next statement. Any REXX statement can be typed or a variable can be modified before processing the enter key.

7 Interactive Debug Facility

The interactive debug has the following features:

- C Control execution of a REXX program
- C View tracing of execution separated by pauses
- C Insert instruction at a breakpoint.
- C Re-execute previous instruction

To start interactive debugging issue a:

TRACE with the ? option (ie: TRACE ?R)

To continue tracing across procedures issue a:

EXECUTIL TS

To terminate interactive trace use TRACE off. The EXECUTIL TE also terminates interactive debugging.

TS or TE can be issued from an attention interrupt by (PAI) after the following is displayed.

ENTER HI TO END, A NULL LINE TO CONTINUE, OR AN IMMEDIATE COMMAND

Interactive Trace Options at a Pause

- C Continue with null line (Enter key).
- C Additional Commands.
- C Equal sign (=) re executes the previous command.

8 TSO/E External Functions - LISTDSI

```
LISTDSI (dsn, [DIRECTORY | NODIRECTORY] )
```

This command returns attributes and characteristics of a data set. The attributes are returned into a set of predefined variables. If used as a function, it returns a code indicating the status of the command.

The possible values are:

0	Normal Completion
4	Some Data Unavailable
16	Error detected

Variable Names and Contents

Variable Name	Contents
SYSDSNAME	Data Set Name
SYSVOLUME	Volume
SYSUNIT	Device Unit
SYSDSORG	Organization (PS,DA,PO,VS)
SYSRECFM	Record Format (U,F,V,S,B)
SYSLRECL	Record Length
SYSBLKSIZE	Block Size
SYSALLOC	Allocation in space units
SYSUSED	Allocation used in space units
SYSPRIMARY	Primary allocation
SYSSECONDS	Secondary allocation
SYSUNITS	Space Units
SYSEXTENTS	Extents Used
SYSCREATE	Creation Date

9 OUTTRAP Function

```
OUTTRAP (OFF)
```

```
OUTTRAP (varname, [maxlines] [, CONCAT | NOCONCAT] )
```

- C The OUTTRAP captures command output and places them into a series of numbered variables (compound variables).
- C The varname is the stem of the compound variable.
- C If maxlines is omitted, the default is all lines.
- C CONCAT appends all command output.

For NOCONCAT each command starts at the beginning.

Special variables are available after the OUTTRAP function is executed:

varname.0	Largest index which output was trapped.
varname.MAX	Maximum number of lines that can be trapped.

9.1 OUTTRAP Example

```
X= OUTTRAP ('MEMB.')
```

```
"LISTDS SAMPLE.DATA MEMBERS"
```

```
DO I = MEMB.0 TO 6 BY-1
```

```
  SAY MEMB.I
```

```
END
```

```
Y = OUTTRAP ('OFF')
```

10 List a Substitute

```
ARG DDNAME
NULL = OUTTRAP('LDD.')
```

"LISTA STATUS"
/* look for DDNAME */
DO I = 1 TO LDD.0 BY 2

```
    IF SUBSTR(LDD.I,1,2)      = ",
    & WORDS(LDD.I)          = 2 ,
    & STRIP(WORD(LDD.I,1))  = DDNAME
    THEN LEAVE I
    ELSE NOP
```

```
END
/* TERMINATE IF DDNAME IS NOT FOUND */
IF I > LDD.0 THEN
    DO
        SAY 'DDNAME IS NOT ALLOCATED'
    END
```

/* ddname is found look for next ddname */

```
    Do J= I+2 to LDD.0 BY 2

    IF SUBSTR(LDD.J,1,2) = "
    & WORDS (LDD.J) = 2
    THEN LEAVE
    ELSE NOP
    END
```

```
IF J > LDD.0
    THEN J=J+1 /* adjust for last entry */
CNT= (J-1)/2
SAY CNT "datasets allocated to "DDNAME
/* DISPLAY the DDNAMES */
DO K = I-1 TO J-2 BY 2
    SAY LDD.K
END
```

11 SYSDSN Function

SYSDSN (dsn)

- C This function returns a message indicating whether a data set exists.
 - C The dsn can be the name of a sequential or PDS data set.
-

RETURNED MESSAGES

The following are the returned messages:

OK
MEMBER NOT FOUND
MEMBER SPECIFIED, BUT DATASET NOT PARTITIONED
DATASET NOT FOUND
ERROR PROCESSING REQUEST DATASET
PROTECTED DATASET
VOLUME NOT ON SYSTEM
INVALID DATA SET NAME
MISSING DATA SET NAME
UNAVAILABLE DATA SET

Note: A fully qualified data set name must be in quotes.

Example:

```
X = SYSDSN("PTC001.CLASS.DATA")
```

12 SYSVAR Function

SYSVAR(info_field)

User	SYSPREF	User prefix
	SYSORIC	Logon PROC
	SYSUID	Userid

Terminal	SYSLTERM	Terminal lines
	SYSWTERM	Terminal width

Execution	SYSENV	FORE or BACK
	SYSICMD	Implicitly executed exec name
	SYSISPF	ACTIVE or NOT ACTIVE
	SYSNEST	YES or NO
	SYSPCMD	Previous command

System	SYSCPU	CPU time during session
	SYSLRECF	RACF Level
	SYSRACF	AVAILABLE, NOT AVAILABLE or NOT INSTALLED
	SYSSRV	System Resource Manager (SRM)
	SYSPCMD	Previous command
	SYSTOE	Level of TSO

13 Using the Data Stack and Queue

STACK:

- C A STACK is a structure that can only be accessed from the top.
- C Items can be 'pushed' onto a stack or items can be 'pulled' off of a stack.

NOTE: The item you 'PULL' off of the stack will always be the last item that was 'PUSHED' onto the stack.

QUEUES:

- C A QUEUE is a structure that can be accessed from both ends.
- C Items can be 'QUEUED' onto the bottom of the queue, or items can be 'PULLED' from the top of the queue.

14 Using Buffers

- C New BUFFERS can be created using the MAKEBUF command.
- C QUEUE, PUSH, and their equivalents put data into the last buffer created.
- C Set up your own buffer in the program stack by using the MAKEBUF command.
- C Find out how many entries are already on the stack by using the QUEUED () function.
- C Put data onto the program stack by using the QUEUED instruction or an equivalent command.
- C Remove data from the program stack by using the PULL instruction or an equivalent command.
- C To avoid removing items from the program stack that do not belong to you, remove items one at a time. Prior to removing the items, check to be sure that what you are about to remove is yours.
- C Use the DROPBUF command for ensuring that all of your data has been removed from the program stack before you return to TSO.

15 QUEUE Instruction

- C The QUEUE instruction will place data at the bottom of the TSO program stack.

```
QUEUE expression
```

- C Where expression is the data being placed on the stack.
-

Example:

```
/* USING QUEUES */  
QUEUE ' line 1 '  
QUEUE ' line 2 '  
QUEUE ' line 3 '  
PULL first line  
SAY firstline  
PULL secondline  
SAY secondline  
PULL thirdline  
SAY thirdline  
EXIT
```

16 PUSH Instruction

- C The PUSH instruction will place data at the top of the TSO program stack.
 - C PUSH expression.
 - C Where expression is the data being placed on the stack.
 - C The REXX QUEUED() function will return the number of lines remaining on the TSO program stack at the time the function is invoked.
 - C symbol = QUEUED()
-

Example:

```
/* USING PUSH */  
PUSH ' line 1 '  
PUSH ' line 2 '  
PUSH ' line 3 '  
PULL firstline  
SAY firstline  
PULL secondline  
SAY secondline  
PULL thirdline  
SAY thirdline  
EXIT
```

17 EMPTY A STACK/QUEUE

```
/*      EMPTY A STACK/QUEUE      */  
NUMB_ELEM = QUEUED ()  
  
DO      NUMB_ELEM  
        PULL ELEMENT  
        SAY ELEMENT  
END
```

- C PARSE EXTERNAL always goes to the terminal instead of checking the data stack first.
- C Stack can have unlimited number of elements.

18 Creating a Buffer on the Data Stack

- C MAKEBUF creates a buffer which you can think of as an extension of the stack.
- C DROPBUF deletes a buffer and all elements in it.
- C The buff allows a REXX program.
- C Queue elements on a stack that already contains a stack.
- C Temporary storage that is easily deleted.

Multiple buffers can be created. RC return the number of buffers after a MAKEBUF.

MAKEBUF does not prevent accessing elements below it.

When an element is removed below the buffer, the buffer disappears.

QBUF returns the number of buffers in RC.

QELEM returns the number of elements in most recent buffer and puts it in RC.

19 Preventing Access to Elements Below

```
/*      Preventing access to elements below      */  
  
NUMBER_OLDQ = QUEUED()  
  
'MAKEBUF'  
PUSH      'ABC'  
QUEUE     'XYZ'  
  
DO WHILE QUEUED () > NUMBER_OLDQ  
    _____  
  
END  
  
DROPBUF
```

20 I/O in REXX

```
EXECIO [lines|*] DISKW ddname
      ([STEM varname]
```

```
EXECIO [lines|*] DISKR ddname starting-record
      ([STEM varname] [FINIS] [LIFO|FIFO|SKIP])
```

```
EXECIO [lines|*] DISKRU ddname starting-record
      ([STEM varname] [FINIS] [LIFO|FIFO|SKIP])
```

- C These commands are used to write, read and read with update a sequential data set.
- C The DISKR/DISKRU commands read the record into the stack unless the STEM option is requested.
- C The files must be allocated before the EXECIO commands can be executed.

The following values are returned:

0	Normal
1	Data Truncated
2	End Of File
20	Severe Error

Example:

```
"ALLOC DA(SAMPLE.DATA) DDN(INDD) SHR
```

```
"ALLOC DA(NEW.DATA) DDN(OUTDD) NEW ...."
```

```
SAY 'Copying...'
```

```
"EXECIO * DISKR indd (FINIS"
```

```
QUEUE "
```

```
"EXECIO * DISKw indd (FINIS"
```

```
SAY 'Copy Complete'
```

```
"FREE DDN(indd outdd)"
```

C FINIS - will close the dataset when done

21 Invoking an EXEC from an EXEC

A REXX program with another program can be invoked explicitly by coding the member name.

Passing arguments is the same as running any REXX program. Optionally, a value can be returned with the RETURN or EXIT command. When control is returned, the value is returned in RESULT.

22 Host Command Environment

TSO	xTSO/E address space.
z/OS	Non TSO/E address space.
LINK	Link to a z/OS module on the same task level.
ATTACH	Attaches to a z/OS module or a different task level.
ISPEXEC	Run Dialog Manager commands.
ISREDIT	Run Edit Macro.

23 Changing Host Environment

```
ADDRESS environment [single command]
```

Example:

```
ADDRESS ISPEXEC "EDIT DATASET("DSN")"
```

To determine the host environment, use the ADDRESS function.

```
ENV = ADDRESS()
```

The ADDRESS function returns one of the following:

TSO

z/OS

LINK

ATTACH

ISPEXEC

ISREDIT

24 SUBCOM Command

SUBCOM environment

The SUBCOM command is used to determine if a host command environment is available.

If the environment is present, RC contains a 0 - otherwise it returns a 1.

Example:

```
SUBCOM ISPEXEC  
If RC = 0 THEN  
  ADDRESS "ISPEXEC EDIT DATASET(STUFF.DATA)"  
ELSE  
  "EDIT" STUFF.DATA
```

**25 REXX Components in
Non-TSO Address Space**

- C All REXX keyword instructions
- C All built-in functions
- C External function STORAGE

TSO REXX Commands

MAKEBUF

DROPBUF

NEWSTACK

DELSTACK

QBUF

QELEM

QSTACK

EXECIO

TS

TE

SUBCOM

26 Executing REXX in TSO Address Space

```
//REXXJCL EXEC PGM=IKJEFT01,  
// DYNAMNBR=30,REGION=4096K  
  
//SYSEXEC DD DSN=uid.REXX.EXEC,DISP=SHR  
  
//SYSTSPRT DD SYSOUT=*  
  
//SYSTSIN DD *  
  
    EXECUTIL SEARCHDD(YES)  
  
    EXEC MEMBER EXEC  
/*
```

DYNAMNBR = 30

Allows TSO to dynamically allocate DD's (30) needed at execution time.

27 Program IRXJCL

```
//STEP01 EXEC PGM=IRXJCL,PARM='execname parm1 parm2 . . .'
```

The above JCL command executes a REXX exec in a non-TSO (z/OS) environment.

To invoke another exec from within z/OS, issue the following:

```
ADDRESS z/OS "execname parm1 parm2..."
```

```
ADDRESS z/OS "EXEC execname parm1 parm2..."
```

28 ALTLIB (z/OS Features)

- C The ALTLIB command activates or deactivates REXX libraries for implicit execution.
- C ALTLIB can also be used for CLIST.
- C Alternate libraries can be set by user, application and system.

User Level

Search SYSUEXEC and SYSUPROC first.

Application Level

Specified by data set name.

System Level

Instead of SYSEXEC or SYSPROC use another library name.

ALTLIB Operands

ACTIVATE

DEACTIVATE

DISPLAY

RESET

29 List a Substitute

```

/*REXX*/
@tcb = STORAGE(21C,4)          /* get TCB address from PSAALD */
@tiot = STORAGE(DSX(C2D(@tcb)+12),4) /* add 12 to get TIOT address */
@1stdd = D2X(C2D(@tiot)+28)    /* point at 1st DD in TIOT */
@1stddlen = STORAGE(D2X(@tiot)+24),1) /* get its length */
@tiot = D2X(C2D(@tiot)+24)    /* point at 1st TIOT DDNAME */
@jfcbb = STORAGE(D2X(X2D(@tiot)+12),3) /* get JFCB address from TIOT */
Say ' DDNAME VOLSER DATA SET NAME'
Say '-----'
Do While C2(@1stddlen) \=0     /* loop through entire TIOT */
  Say STORAGE(@1stdd,8),      /* display DDNAME */
    STORAGE(D2X(C2D@jfcbb)+134),6), /* display volser */
    STORAGE(D2X(C2D(@jfcbb)+16),44) /* display DSN */
  @tiot = D2(X2D(@tiot)+C@(@1stddlen)) /* point to next TIOT entry */
  @1stddlen = STORAGE(@tiot,1) /* get length of TIOT entry */
  @1stdd = D2(X2D(@tiot)+4) /* point at DDNAME in TIOT */
  @jfcbb = STORAGE(D2X(X2D(@tiot)+12),3) /* get net JFCB address */

```