REXX

REstructured e**X**ecutor e**X**tended language is a highly versatile programming language, introduced in 1983. A person who is just FAMILIAR with REXX can do lot of valuable automation in any project that has repeated-manual tasks. What I am covering here on REXX is only 10 percent of it and that too in a refreshing way.

SYSPROC and SYSEXEC

TSO ISRDDN lists all the datasets allocated for the your session. The datasets that are concatenated under SYSPROC are CLIST and SYSEXEC are REXX. This is just a convention. REXX can also be placed in SYSPROC libraries. In such cases, to differentiate REXX from CLIST, the first statement of the member should contain the keyword REXX. (/*REXX*/)

Whenever a utility is invoked, the order of search is- SYSUEXEC, SYSEXEC, SYSPROC.

Executing your REXX program

The REXX coded in 'BPMAIN.TEST.UTILS(FST)' can be executed in following three ways.

Way 1:

TSO EX 'BPMAIN.TEST.UTILS(FST)' will execute the utility 'FST'.
This is TSO execution command. So if you execute it from option 6 (TSO Panel), TSO keyword is not necessary.

Way 2:

Add your REXX PDS to SYSUEXEC/SYSEXEC/SYSPROC. Then the utility can be directly invoked by just mentioning the utility name.
"ALLOC FI(SYSUEXEC) DA('BPMAIN.TEST.UTILS') SHR"
"ALTLIB ACTIVATE USER(EXEC)"

The first command allocates the PDS to SYSUEXEC and the second command activates it. The addition of the PDS to SYSUEXEC can be crosschecked using ALTLIB DISPLAY and ISRDDN commands.

Way 3:

When there is a requirement to execute the REXX in batch, use IKJEFT01 with SYSTSIN card as EX 'BPMAIN.TEST.UTILS(FST)'.

Host environment commands

REXX can be intermixed with commands to different host environments. Host environment commands are identified within double quotes. By default, the commands coded between the double quotes are routed to TSO environment.
REXX can talk to almost any environment. The host environment can be changed by the command ADDRESS.
ADDRESS ISPEXEC changes the environment to ISPEXEC.

Variable, Stem and Queue

Variables are implicitly defined. There is no need to declare/define them. Any word can be used as variable except RESULT, RC and SIGNAL.

STEM is like an array in other languages. But the occurrences of stem can be of different type or length. We can call them more precisely as compound variables.

File can be read into either STEM or QUEUE. Random access is possible with stems and I have explained all the I-O commands with respect to STEMS.

File Operations

Allocate the file to your REXX
"ALLOC FI(FILE1) DA('SMSXL86.SRCE(INP)') SHR "

To read all the records of the file:
"EXECIO * DISKR FILE1 (STEM FILE1. FINIS"
To read N number of records from Mth record:
"EXECIO N DISKR FILE1 M (STEM FILE1. FINIS"
To write all the records of the array to a file.
"EXECIO * DISKW FILE1 (STEM FILE1. FINIS"
To write only N records of the array to a file.
"EXECIO N DISKW FILE1 (STEM FILE. FINIS"

De-allocate the file:
FREE FI(FILE1)

FILE1.0 importance:
Every READ I-O will return the number of records retrieved in stem. 0
In the first/second cases, FILE1.0 contains the number of records retrieved. FILE1.1 contains first record and FILE1.N contains Nth record.

SAY-PULL-EXIT-SIGNAL

Display variable VAR1
SAY 'VALUE OF VAR1 IS:' VAR1
To get the value from user
PULL VAR1 (Stack should empty- or it will be taken from stack)
To come out of REXX program
EXIT [RETURN-CODE]
To transfer the control unconditionally
SIGNAL ON ERROR
…
ERROR: EXIT 12

OUTTRAP-MSG-SLEEP

Output of TSO command execution can be trapped into an array using OUTTRAP command.

X=OUTTRAP(T.)
'STATUS JOBNAME'
X=OUTTRAP("OFF")
T.1 contains the status of JOBNAME.

TSO Messages can be disabled using MSG command.
STATUS=MSG('OFF/ON') – To OFF and ON
STATUS=MSG () – To get the current status.

To introduce N seconds delay.

```
IF SYSCALLS("ON") = 0 THEN
      ADDRESS SYSCALL "SLEEP" N
```

Do Statements

Loop- when occurrence is not needed inside

```
DO count
END
```

Loop- when the occurrence is needed inside

```
DO variable= from-exp TO to-exp [BY exp]
     UNTIL/WHILE expression
END
```

WHILE – Condition check before execution
UNTIL – Condition check after execution
Loop-Forever

```
DO FOREVER
    IF some-condition-met THEN LEAVE
END
```

If LEAVE is not there then program will get into infinite loop.
ITERATE is used for bypassing only one occurrence.

Example:

```
DO I =1 TO 10
    IF I=2 THEN ITERATE I
     IF I=4 THEN LEAVE I
     SAY I
END
```

Result: 1 3
For 2.loop of I is iterated and for 4, loop of I is exited.

IF Statement

```
IF condition-1 THEN
   DO
   END
ELSE IF condition-2 THEN
        DO
           IF condition-3 THEN
              DO
              END
        END
 END
```

When you execute more than one statement in an IF path, don't forget to provide DO-END cover. NOP is no-operation-statement.
-Can be used as CONTINUE in COBOL

SELECT statement

```
SELECT
WHEN condition(s) THEN
  DO
    Statements
  END
WHEN condition(s) THEN
   DO
     Statements
   END
OTHERWISE
      Statements
END
```

Powerful parsing in REXX

Syntax:
PARSE [UPPER] {ARG|EXTERNAL|PULL|VAR| VALUE expression WITH} template.

VAR and VALUE

To parse a value in a variable, use PARSE VAR.
To parse a value, use PARSE VALUE WITH.
WITH clause is mandatory with VALUE to identify end of string.
PARSE VALUE 'Mainframe Refresher Part-1 ' WITH STR1 STR2
Result:
STR1 Mainframe STR2 Refresher Part-1

PULL and EXTERNAL

EXTERNAL parses the string received from the terminal. PULL parses the string in the stack. If stack is empty, it parses the string from TERMINAL.

ARG and PARSE ARG

ARG is used to parse the arguments passed to the program /function/ procedure. PARSE ARG does the same but there will NOT be any upper case conversion of the passed string.
PARSE UPPER ARG and ARG are the same.

Samples:
String = 'Mainframe Refresher Part-1'
PARSE VAR String STR1 STR2 STR3
STR1 Mainframe STR2 Refresher STR3- Part-1

PARSE VAR String STR1 STR2
STR1 Mainframe STR2 Refresher Part-1

PARSE VAR String STR1 '-' STR2
STR1 Mainframe Refresher Part STR2 1

PARSE VAR String STR1 5 10 STR2
STR1 Main STR2 Refresher Part-1

PARSE VAR String with STR1 +4 1 STR2 +3
STR-1 Main STR-2 Mai

Subroutines and Functions

Subroutines are invoked using CALL statements, Functions are invoked in a variable assignment statement.

CALL SUB1 executes procedure SUB1.
X=SUB1 executes the function SUB1.

Procedure may or may not return a value whereas a function always returns a value.
If procedure returns any value, then that is stored in a special register called RESULT.

CALL PARA-1
Other statements
EXIT.

PARA-1:
Statements
RETURN.
PARA-1 can be invoked anywhere above it. After the execution, control will return back to next statement following invoking PARA-1. Last statement of paragraph should be RETURN.

In the above case, all the variables of main paragraph are available to PARA-1 and any variable in PARA-1 is available outside PARA-1.

If PARA-1 is suffixed with PROCEDURE keyword, then all the variables in PARA-1 are local to that procedure.
PARA-1: PROCEDURE

If you want to expose only selected variables of PARA-1 to main exec and main exec to PARA-1, then use EXPOSE along with PROCEDURE.

PARA-1:PROCEDURE EXPOSE VAR1
Here VAR1 is shared by main exec and PARA-1.

Operators

>, <, >=, <=	Same meaning as in other languages.
>>, <<. ==	Strict operators. 'YES' ='Yes' is true but 'YES' == 'Yes' is false.
|, &, &&	OR, AND, XOR respectively
||	Concatenation without space
* / + - = **	Same meaning as in other languages
//	Returns remainder. (10//3 = 1)
%	Divide and return the whole number. (7%2 = 3)

REXX Built-in Functions

Function	Purpose
STRIP(String,Option,Char)	Deletes spaces (or char) from string based on the value specified for option. L- leading, T- Trailing, B- L and T
SUBSTR(string,n,len)	Returns sub-string of length len extracted from string beginning at position n.
POS(string1,string2,start)	Returns the character position of the first occurrence of string1 in string2. If specified, the search begins at location start. INDEX function can also be used for this purpose.
LENGTH(String)	Returns the length of string
USERID()	Returns the TSO user-id
DATE()	Returns the current date in the format DD Month CCYY. DATE(x), x can be B C D E J M N O S U W for different formats.
TIME()	Returns the current time in the format HH:MM:SS TIME(y), y can be C E H L M N R S for different formats.
DATATYPE(string,type)	Returns 1 if string belongs to type else 0. Types can be A,B,D,L,M,N,S,U,W and X. If type is not specified then the result will be NUM or CHAR based on content of string.
INSERT (string1,string2,n,len)	Inserts string1 into string2 at the character location n. If len is specified, string1 is padded to the specified length.
COMPARE(string1,string2)	If string1 and string2 are the same, returns 0; Otherwise, returns the position of the first character that differs.
FORMAT(number, Before,after)	Returns a string that contains a formatted number with before characters on the left of the decimal point and after characters on the right of the decimal point.
OVERLAY(string1,string2,n, len)	Replaces len characters in string2, starting at position n, with the characters in string1. The default for len is the length of string1.
TRANSLATE(string,table1, Table2)	Translates the characters in string found in table2 to the corresponding characters in table1. If both tables are omitted, translates string to uppercase.
TRUNC(number,n)	Returns number with the numbers right of the decimal point truncated to n characters. Default value of n is 0.
DELSTR(string,n,len)	Deletes len characters from string beginning at position n. If len is omitted, the rest of the string is deleted.
LEFT(string1,len)	Returns a string of length len where string1 is left justified. RIGHT and CENTER are used for right and center justification.
SYSDSN('data-set-name')	Returns the status of dataset. OK if it is accessible.

Other functions are: ABBREV, ABS, ADDRESS, BITXOR,COPIES,C2D,C2X,X2D, X2C,D2C,D2X,WORD,WORDINDEX,DELWORD,WORLENGTH,WORDPOS,WORDS, SUBWORD,REVERSE,SIGN,MAX,MIN,RANDOM,SPACE,SYMBOL,VERIFY,FUZZ, LASTPOS,LINESIZE,SOURCELINE,FORM,COPIES,ERRORTEXT,EXTERNALS,DIGITS, CONDITION,JUSTIFY,SYMBOL, ARG,BITAND,BITOR

NUMERIC DIGITS:

You can set the precision of numeric operation to any level you wish. (Default is 9 digits)

To change the precision to 100 digits : NUMERIC DIGITS 100

INTERPRET

1. If you want to execute a command that is in a variable, you can do it using INTERPRET. VAR='A=A+1', INTERPRET VAR adds 1 to a.
2. If you want to get a variable name from user and assign value to it, you can do it using INTERPRET. PULL VAR1, INTERPRET VAR1 '=' 10 => assigns the value 10 to the variable extracted from user.

PUSH-PULL-QUEUE and QUEUED

PUSH and QUEUE are used to store the elements in a stack. PULL is used to pull off the elements from a stack. QUEUED elements are pulled off in FIFO basis. (First in First Out) whereas PUSHED elements are pulled off in LIFO bases. (Last in First Out basis). When you have done mixture of QUEUE and PUSH, the PUSHED items will appear first. QUEUED() gets the number of elements in a stack.

SIGNAL

It is equivalent to GOTO of other languages. SIGNAL PARA-1 unconditionally transfers the control to the label PARA-1. This command is mainly used for error handling.

SIGNAL command can trap the following conditions.

SYNTAX –Any syntax error
ERROR -Any environment command that results a non-zero return code.
HALT -When the user interrupts execution
NOVALUE -When a symbol is used without having been given a value

Syntax: SIGNAL ON <one-of-the–above-condition>

There should be a label in your REXX with the condition name. You can give your own name for error routines using 'SIGNAL ON <condition> name para-name'.

Ex:

```
SIGNAL ON ERROR          | SIGNAL ON ERROR NAME ERROR-PARA
…                        | …..
ERROR: EXIT 12           | ERROR-PARA: EXIT 12
```

Handled conditions can be deactivated using SIGNAL OFF <handled-condition> .

TRACING

If a program goes wrong and you need more information in order to work out why, then REXX provides you with the ability to trace all or part of your program to see what is happening.

The most common form of tracing is turned on by the command, TRACE R

This causes each instruction to be listed before it is executed. It displays the results of each calculation after it has been found.

Another useful trace command is TRACE ?A

This makes the interpreter list the next instruction and then stop.

You can continue to the next instruction by pressing return, or you can type in some REXX to be interpreted, after which the interpreter will pause again. You can use this to examine the program's variables and so forth.

TSO COMMANDS:

Frequently used TSO commands are listed in the table. They can be issued from any ISPF panel with TSO prefix. TSO prefix is not required if you type them in TSO panel. (Option 6 of ISPF)

If the command is incomplete, TSO prompts for the missed parameters. The output of the processed command is displayed in ISPF, in line mode. If you issue TSO commands in REXX, capture the output into an array using OUTTRAP command and later read the array to get the required information.

For example, if you want to process/change all the members of a PDS, the process would be:

Issue OUTTRAP command – LISTDS on the PDS with MEMBERS option – Deactivate the OUTTRAP command - Read the array that is loaded with LISTDS output – MEMBER NAMES are stored from 7th element – For every read after 7th occurrence, you will get one member name– Concatenate this member name with PDS and allocate the PDS member for your REXX – Read the allocated member into array/stack – Process/Change as per requirement – Write back into PDS if needed – Read the next member in array and do the process in loop.

Command Syntax	Purpose
HELP [COMMAND]	Displays the purpose and syntax of the command. The source for this information is retrieved from 'SYS1.HELP'. Ex: HELP ALLOCATE
SEND	It is used to communicate with other users. SEND 'hi' USER(SMSXL86) LOGON Message 'hi' is sent to user SMSXL86. If the user is not logged on, the message should be displayed the next time user logs on. LOGON is an optional parameter. Message: 115 chars maximum
LISTCAT	It is used to list entries in MVS catalog. The syntax and the available options are already explained in VSAM - IDCAMS section. LISTCAT ENTRIES'SMSXL86.TEST.SOURCE' ALL
LISTDS	It is used get the information about one or more datasets. LISTDS 'dataset-name' MEMBERS|HISTORY|STATUS|LEVEL MEMBERS list all the members of a PDS. This command is useful in REXX to process all the members of a PDS.
RENAME	It is used to rename a dataset. Generic datasets are allowed. RENAME 'BPAMAIN.TEST.*' 'BPMAIN.UNIT.*' renames all the datasets start with BPMAIN.TEST to BPMAIN.UNIT. RENAME 'SMSXL86.TEST(MEM1)' 'MEM2' renames mem1 as mem2 in the PDS 'SMSXL86.TEST'.
DELETE	It is used to delete dataset(s). PURGE qualifier is needed to bypass expiration date checking. DELETE 'BPMAIN.TEST.*' PURGE

ALLOCATE (Existing dataset)	ALLOCATE DA('SMSXL86.TEST.SOURCE') FILE(INP) SHR It allocates the dataset 'SMSXL86.TEST.SOURCE' with logical name INP. INP can be used in REXX for READ/WRITE.
ALLOCATE (New dataset)	Allocation can be done using model-dataset attributes or with new attributes or combination of both. Attributes that can be specified are: UNIT, VOLUME, SPACE, TRACKS|CYLINDERS|BLOCK(blk-length), DIR, RECFM, LRECL, BLKSIZE, DATACLASS, STORCLAS, MGMTCLAS, EXPDT/RETPD. Model parameter: LIKE('model-dataset')
FREE	De-allocate the allocated-datasets. FREE FI(logical-name) FREE DSNAME('dataset-name') FREE ALL
ISRDDN	It displays the files/datasets allocated to your session. 'M member' command searches the member in all the datasets allocated to you. If you want to see the source of a utility, give a search using the command ' M utility-name' in the ISRDDN panel.
CALL	It is used to execute a load module. CALL 'BPMAIN.TEST.LOADLIB(TEST1)' – Executes the load TEST1. If the program needs any datasets, then they should be allocated before issuing this command.
SUBMIT	It is used to submit a job to JES. SUBMIT 'PDS(MEMBER)'[JOBCHAR('A')]
CANCEL	It is used to CANCEL /PURGE the job from JES. CANCEL JOB-NAME| JOB-NAME(JOB-ID) [PURGE]
STATUS	It is used to get the status of the submitted-job in JES. STATUS JOB-NAME | JOB-NAME(JOB-ID)

MACROS:

An edit macro is a REXX/CLIST procedure that you can invoke from the ISPF editor as if it were an ISPF primary command. Sequence of ISPF commands can be stored as a part of macro and this can be invoked as and when needed. Macros are stored in SYSEXEC or SYSPROC libraries.

The first ISREDIT command in an edit macro must be ISREDIT MACRO statement. MACRO statement can optionally receives arguments. Here is a simple edit macro that search for a specific string passed to the macro:

```
ADDRESS ISREDIT
"MACRO (STRING)"          => It expects an argument on execution
"EXCLUDE ALL"             => Excludes all the lines in the member
"FIND" STRING "ALL"       => Searches the passed-argument in the member
```

Store this macro with any name, say FALL.
Open a member in edit mode and type FALL TEST; this will exclude all the lines other than the lines having the string 'TEST'.

COMMANDS IN MACRO

DELETE:

```
Type: 1          DELETE [ALL] [range] [X|NX]
Type: 2          DELETE line_pointer
```

FIND and SEEK:

The syntax of FIND command is same as in ISPF. SEEK is special macro command. When a FIND command finds the search string in an excluded line, it resets the line so it is no longer excluded. As a result, the user can see the line. When a SEEK command finds the search string in an excluded line, the line is not reset, so the user cannot see it.

```
SEEK String [range] {NEXT|PREV|FIRST|LAST|ALL}
                    {CHARS|PREFIX|SUFFIX|WORD}
                     [X|NX] [col-1 [col-2]]
```

range is – range of lines by two labels – default range is the first and last lines of the file.
Col-1 is the starting column of the string. If col-2 is not specified, string must begin in this column. If col-2 is specified, string must be found between col-1 and col-2 to satisfy the search.
Meaning of other keywords is same as ISPF FIND command.

LINE:

It is used to retrieve/set the values in a line.

```
(variable) = LINE line-pointer   => Variable retrieves the data from the line
                                    indicated by line_pointer.

LINE line_pointer = value        => Line indicated by line_pointer is set to the
                                    value coded on the right side.
```

LINENUM:

```
ISREDIT (variable) = LINENUM label =>Variable receives the line number of the line
                                     pointed by label.
```

INSERT:

INSERT line-pointer [count]
'Count' lines are inserted after the line pointed by line_pointer.

Lines are inserted using the above command; to load the lines with data, the following command is used;

{LINE_BEFORE|LINE_AFTER} line-pointer =
{DATALINE|INFOLINE|MSGLINE|NOTELINE}
{value|LINE|line_pointer| MASKLINE|TABSLINE}

1. Before or after the line indicated by line-pointer, either DATALINE or INFOLINE|MSGLINE|NOTELINE is added. The value is mentioned by the next parameter of right hand side.
   DATALINE indicates line of data and it can be stored. Others cannot be stored.
   MSGLINE is identified by ==MSG> in line command area;
   NOTELINE is identified by =NOTE== in line command area;
   INFOLINE is identified by ====== in line command area;

2. VALUE specifies variable or literal string that should be written into this line.
   LINE specifies that data from preceding line is used for the inserted line.
   Line_pointer specifies that data from the specified line is used for the inserted line.
   MASKLINE and TABLINE specify that data from MASKLINE and TABSLINE are used for the inserted line respectively.

LABEL:

Most of the commands listed above use line pointer for identifying a line. The line pointer can be a label or relative line number. Labels are preferred over relative line numbers, as their position will not be modified by deletions or insertions.

There are three predefined variables namely .ZCSR, .ZFIRST and .ZLAST. ZCSR points to current line, ZFIRST points to the first line and ZLAST points to the last line in the member. New labels can be created using the following command:

LABEL line_pointer = .label

Line_pointer is the relative line number or label identifying the line where the label is to be assigned and label is 1-8 character alphabetic name that can start with any letter other than 'Z'.

SHIFT:

SHIFT **(** < | > | (|) | **)** line_pointer [cols]

Line_pointer specifies the line, the line where data is to be shifted.
')' Specifies right shift of characters and '(' Specifies left shift of characters
'>' Specifies right shift of data and '<' Specifies left shift of data.

Default cols is '2'.

ISREDIT MACRO Assignment Statements

Edit assign statements are used to get/edit information about the current editing environment. Some of them can be used only for retrieval and cannot be updated.

(variable) = key_phrase - To get the value
key_phrase = variable - To set the value.

Key_phrase can be:

DATASET	Returns the name of the dataset
MEMBER	Returns the name of the member
CURSOR	Returns two values representing the line and column position of the cursor.
DATA_CHANGED	Returns TRUE is the data has been changed since it was last saved, and NO if it hasn't.
CHANGE_COUNTS	Returns two values- number of strings that were changed and the number of strings that could not be changed.
FIND_COUNTS/ SEEK_COUNTS	Returns two values - the number of strings that were found and the number of lines where strings were found.
EXCLUDE_COUNTS	Returns two values - the number of strings that were found and the number of lines that were excluded.
USER_STATE	Returns the current edit profile in an internal form that can be later stored.
RANGE_CMD	The line command that was used to mark a range.

IMACRO setup

If you want to attach a macro with a file, open the file in edit-mode and type the command IMACRO macro-name. When you save the file, the profile will be updated with this macro name. So the macro will be executed whenever you edit this file.
IMACRO NONE disables the IMACRO already attached to a file.

NOTE:
1. Before issuing ISREDIT commands, you should use ADDRESS command to change the host environment. (ADDRESS ISREDIT)
2. All the host environment commands should be coded with double quotes in the REXX.

How to set error messages in case of errors?

In REXX, it is usual practice to check the RC value after every host environment command. If the value is non-zero, then it has to be handled. If you want to terminate the process and display error message, use the following ISPEXEC command.

For example, if the SEEK command could not find the string, RC will be set to 4. If RC = 4 then you can execute the following set of statements to inform this to the user:

```
ADDRESS ISPEXEC
ZEDSMSG = "STRING NOT FOUND"
ZEDLMSG = "THE STRING ENTERED IS NOT FOUND IN THE MEMBER'
"SETMSG MSG(ISRZ001)"
```

ZEDSMSG is the short message that appears on the right hand-top of the screen.
ZEDLMSG is the long message that appears when the user presses F1 on receiving short message.

Error Codes of ISREDIT MACRO Commands

ISREDIT Command	Return Code	Meaning	ISREDIT Command	Return Code	Meaning
CHANGE	4	String not found	LINE	4	Source data truncated
	8	Unable to make change		8	Variable not found
CREATE	8	Member already exist		12	Invalid line number
DELETE	4	No lines deleted		16	Variable data truncated
	8	No records in file.	LOCATE	4	Line not found
	12	Invalid line number.		8	Empty file
EXCLUDE	4	String not found	PROCESS	4	Default range used
	8	Lines not included		8	Default destination used
FIND	4	String not found		12	Default range and destination used
INSERT	12	Invalid line number		16	Invalid line commands
LABEL	8	Duplicate label replaced	SAVE	4	New member created
	12	Invalid line number		8	Not enough space
LINE_AFTER	4	Data truncated	SEEK	4	String not found
LINE_BEFORE	12	Invalid line number	SHIFT	12	Invalid line number
LINENUM	8	Label not found			

How to allow/restrict others to work on your datasets?

PERMIT command will do this. If I want to protect my user qualified datasets and my ID is MT0012, then the following command will disallow all others access over these datasets.

TSO PERMIT 'MT0012.*' ID(ALL) ACCESS(NONE)

Specific IDs and Specific access can be given in ID/ACCESS parameters Detailed information can be found in SYS1.HELP(PERMIT).

How to do 3.4 search in batch

The following REXX/JCL will do.

```
//STEP002  EXEC PGM=IKJEFT01,DYNAMNBR=128
//ISPPLIB  DD DISP=SHR,DSN=ISP.SISPPENU
//ISPSLIB  DD DISP=SHR,DSN=ISP.SISPSLIB
//         DD DISP=SHR,DSN=ISP.SISPSENU
//ISPMLIB  DD DISP=SHR,DSN=ISP.SISPMENU
//ISPPROF  DD DSN=MT0012.ISPF.TEMPPROF,DISP=SHR
//ISPTLIB  DD DISP=SHR,DSN=ISP.SISPTENU
//ISPLOG   DD SYSOUT=*,
//            DCB=(LRECL=120,BLKSIZE=2400,DSORG=PS,RECFM=FB)
//ISPLIST  DD SYSOUT=*,
//            DCB=(LRECL=121,BLKSIZE=1210,RECFM=FBA)
//SYSEXEC  DD DISP=SHR,DSN=ISP.SISPEXEC
//SYSPROC  DD DISP=SHR,DSN=MT0012.TOOLS
//         DD DISP=SHR,DSN=ISP.SISPCLIB
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
  PROFILE PREFIX(MT0012)
  ISPSTART CMD(%DSLIST M%%P.M%%PFM.COMPRINT.I*)
/*
//OUTPUT  DD DSN=MT0012.DATASETS.COMPRINT,
//           DISP=(NEW,CATLG,DELETE),
//           SPACE=(TRK,(10,10),RLSE),
//           DCB=(LRECL=80,RECFM=FB)
//*
```

LMLIST is a REXX that resides in SYSPROC library(MT0012.TOOLS):

```
ARG DATASET
SAY 'GENERIC DATASET PASSED: ' DATASET

ADDRESS ISPEXEC
"LMDINIT LISTID(LISTID) LEVEL("DATASET")"
SAY 'LISTID                :' L_RC LISTID
DSNAME = " "
DSCOUNT = 0
DO UNTIL RC > 0
  "LMDLIST LISTID("LISTID") DATASET(DSNAME) OPTION(LIST) STATS(NO)"
  IF RC = 0 THEN DO
    DSCOUNT = DSCOUNT + 1
    DSN.DSCOUNT = DSNAME
    SAY 'DATASET FOUND         : ' DSNAME
  END
END
SAY 'TOTAL DATASETS  FOUND : ' DSCOUNT
ADDRESS TSO
```

```
"EXECIO * DISKW OUTPUT (STEM DSN. FINIS"
"FREE FI(OUTPUT)"

"LMDFREE LISTID(LISTID)"

EXIT (0)
```