TSO/E External Functions

Objectives

TSO/E External Functions

In this module, you will look at the TSO/E external functions that enable REXX to use many of the facilities and options in the older CLIST interpreted language.

After completing this module, you will be able to:

- Identify TSO/E External Functions for Interrogating Data Set Information
- Identify TSO/E External Functions for Controlling and Trapping Messages
- Identify TSO/E External Functions for Interrogating System Information

TSO/E external functions

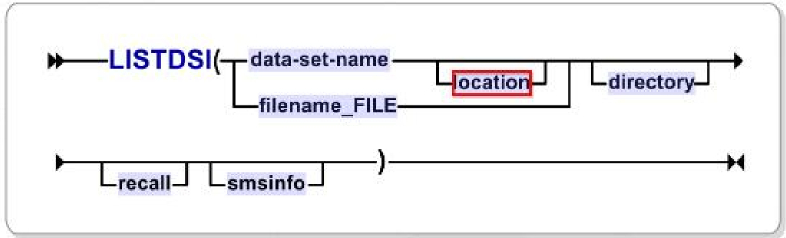
- LISTDSI
- MSG
- MVSVAR
- OUTTRAP
- PROMPT
- STORAGE
- SYSDSN
- SYSVAR

Retrieves information about data sets, such as their allocation status, attributes, protection status, and location.

After REXX was ported to the MVS and TSO/E operating environment in 1988, there were several useful functions in the CLIST interpreted language that REXX could not perform.

These were rewritten as special functions that are available only when REXX is running under TSO/E. They can be called in the same way as other REXX functions, but the values returned by these functions do not follow the same usage patterns of most other functions.

Mouse-over each of the most commonly used TSO/E external functions for a description.



This specifies how the system should search for the data set. VOLUME (serial ID) indicates that the specified volume serial should be searched. PREALLOC specifies that the location of the data set is determined by allocating the data set rather than by a catalog search. If neither is specified, a standard catalog search is performed.

The LISTDSI function enables information about the structure and attributes of a data set to be used in a REXX procedure. Note that no commas are used in the syntax of the LISTDSI function.

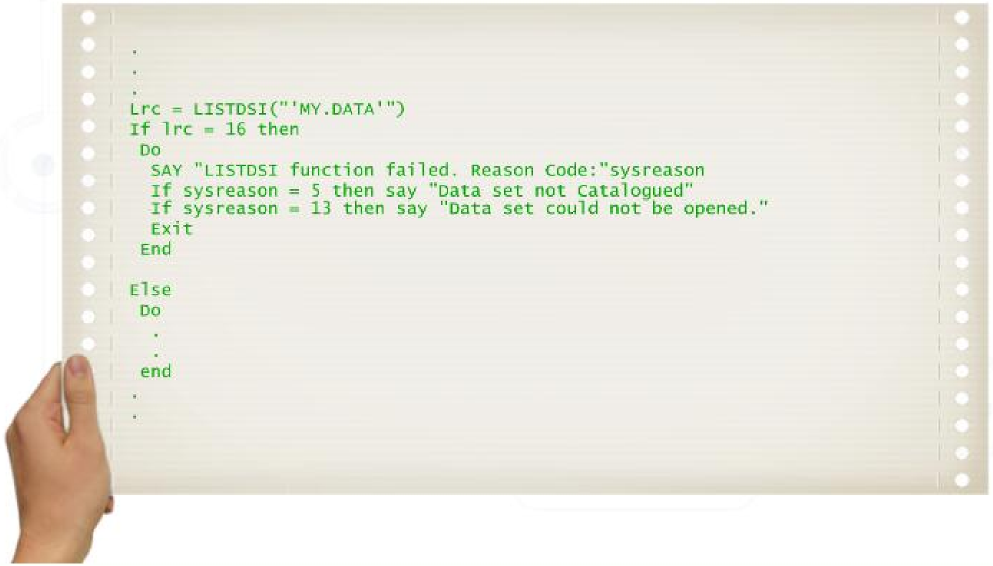
If a data set name is passed without quotes, TSO/E will append the TSO prefix to the beginning of the data set name.

Mouse-over the syntax for a description of each subparameter.

Variables	Description
SYSDATACLASS	The SMS data class name returned only if SMSINFO is specified on the LISTDSI statement and the data set is managed by SMS.
SYSDSNAME	Data set name.
SYSDSORG	Data set organization: **PS** Physical sequential **PSU** Physical sequential unmovable **DA** Direct organization **DAU** Direct organization unmovable **IS** Indexed sequential **ISU** Indexed sequential unmovable **PO** Partitioned organization **POU** Partitioned organization unmovable **VS** VSAM **???** Unknown
SYSDSSMS	Contains information about the type of data set provided by DFSMS/MVS. If the SMSINFO keyword operand on the LISTDSI statement is not specified, or SMS DSNTYPE information

Unlike most built-in functions, the LISTDSI function only returns the return code of the function. Approximately 30 other variables are set and are immediately available in the REXX function variable pool if the return code is 0 or 4.

Scroll through the list of the variables set by the LISTDSI function.

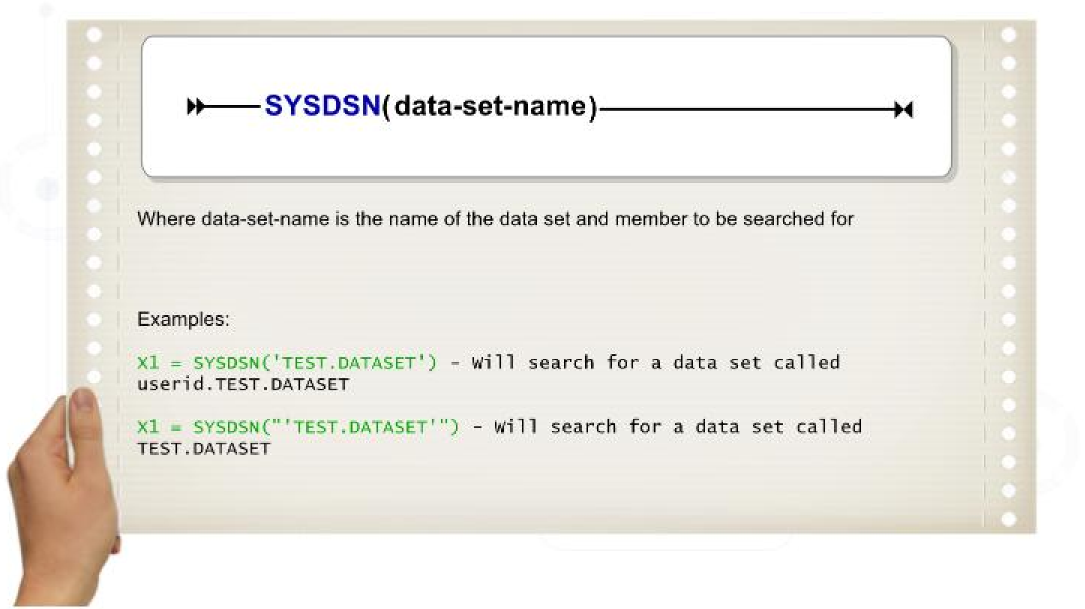


```
.
.
.
Lrc = LISTDSI("'MY.DATA'")
If lrc = 16 then
 Do
  SAY "LISTDSI function failed. Reason Code:"sysreason
  If sysreason = 5 then say "Data set not Catalogued"
  If sysreason = 13 then say "Data set could not be opened."
  Exit
 End

Else
 Do
  .
  .
 end
.
.
```

The three possible values returned by the LISTDSI function, which should always be checked to ensure that the function worked, are:

0 - Successful completion
4 - Successful completion but some values could not be determined
16 - LISTDSI function failed

If LISTDSI returns a value of 16, the variable SYSREASON contains a reason code describing the error. Reason codes are listed in the TSO/E REXX Reference. An example of code that could be used when using the LISTDSI function is shown above.



```
▶▶──SYSDSN(data-set-name)──────────────▶◀
```

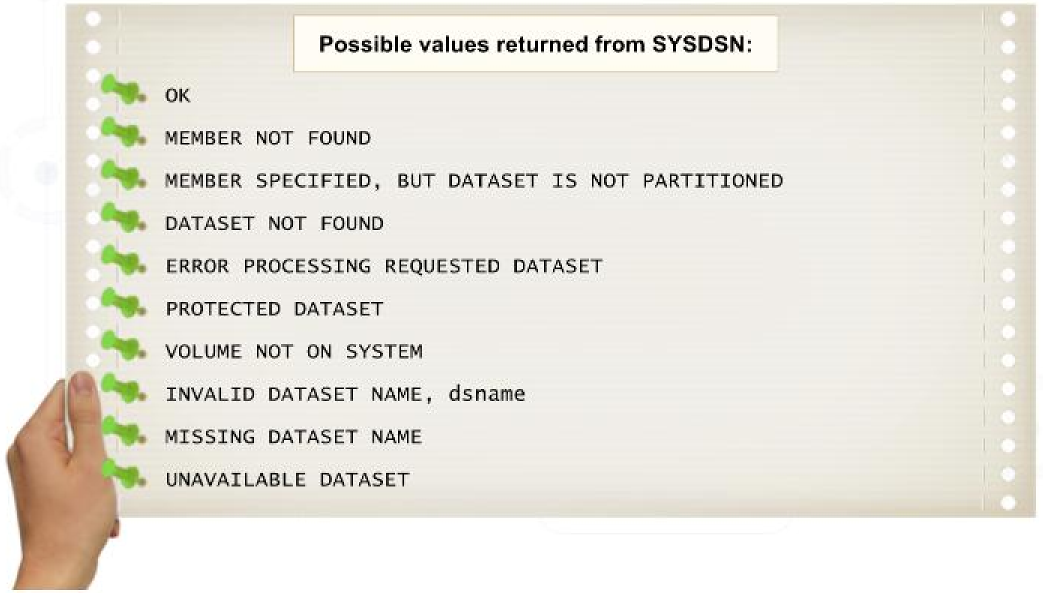
Where data-set-name is the name of the data set and member to be searched for

Examples:

```
X1 = SYSDSN('TEST.DATASET') - Will search for a data set called
userid.TEST.DATASET

X1 = SYSDSN("'TEST.DATASET'") - Will search for a data set called
TEST.DATASET
```

The SYSDSN function can be used to check whether a data set and member exists in the system catalog.

If the data set name is not enclosed in quotes when passed to the system, TSO appends the current TSO prefix to the beginning of the data set name.

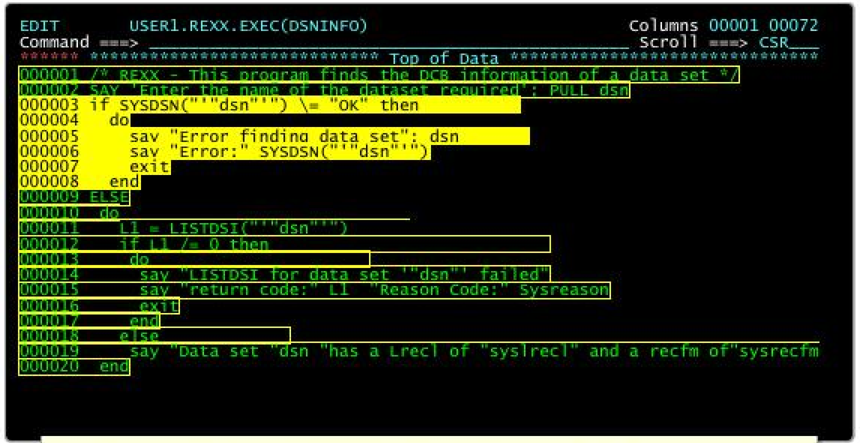


Possible values returned from SYSDSN:

```
OK
MEMBER NOT FOUND
MEMBER SPECIFIED, BUT DATASET IS NOT PARTITIONED
DATASET NOT FOUND
ERROR PROCESSING REQUESTED DATASET
PROTECTED DATASET
VOLUME NOT ON SYSTEM
INVALID DATASET NAME, dsname
MISSING DATASET NAME
UNAVAILABLE DATASET
```

If the SYSDSN function successfully finds a data set in the system catalog, it will return the value OK.

Other values can be returned depending on the error that has occurred. The possible strings that can be returned from the SYSDSN function are listed above.

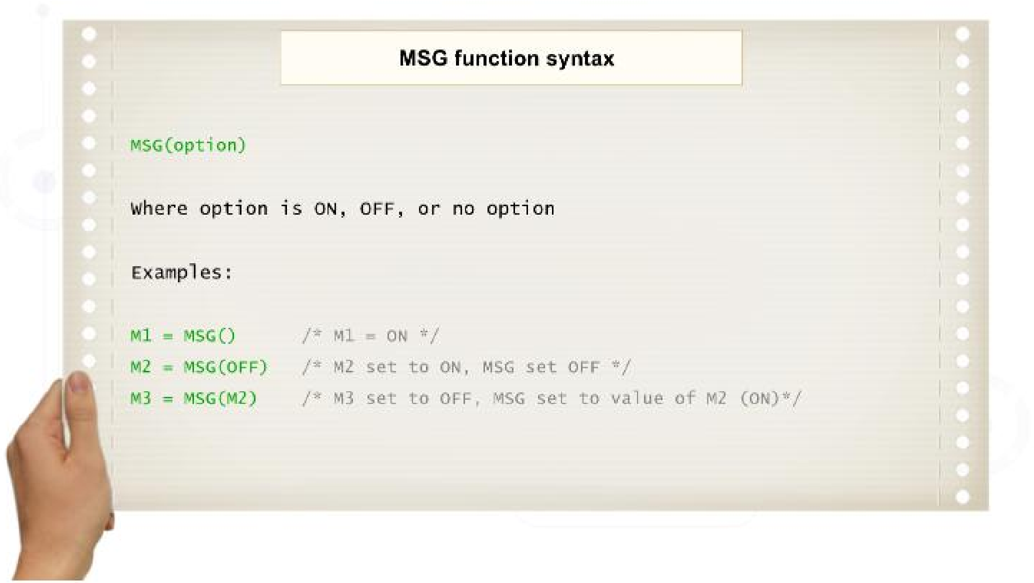


```
EDIT       USER1.REXX.EXEC(DSNINFO)                         Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
/* REXX - This program finds the DCB information of a data set */
SAY 'Enter the name of the dataset required'; PULL dsn
if SYSDSN("'"dsn"'") \= "OK" then
  do
    say "Error finding data set": dsn
    say "Error:" SYSDSN("'"dsn"'")
    exit
  end
ELSE
 do
   L1 = LISTDSI("'"dsn"'")
   if L1 /= 0 then
    do
     say "LISTDSI for data set '"dsn"' failed"
     say "return code:" L1  "Reason Code:" Sysreason
     exit
    end
   else
    say "Data set "dsn "has a Lrecl of "syslrecl" and a recfm of"sysrecfm
 end
```

If the data set exists, the SYSDSN function will return the result "OK". If it does not return "OK", display a message indicating an error occurred, and display the actual returned statement from the SYSDSN function. Note the "'" characters around the dsn variable in the SYSDSN function. These ensure that the data set name is passed with literal single quotes surrounding it.

This is an example of the SYSDSN and LISTDSI functions in use.

Mouse-over the code for a description of each clause.

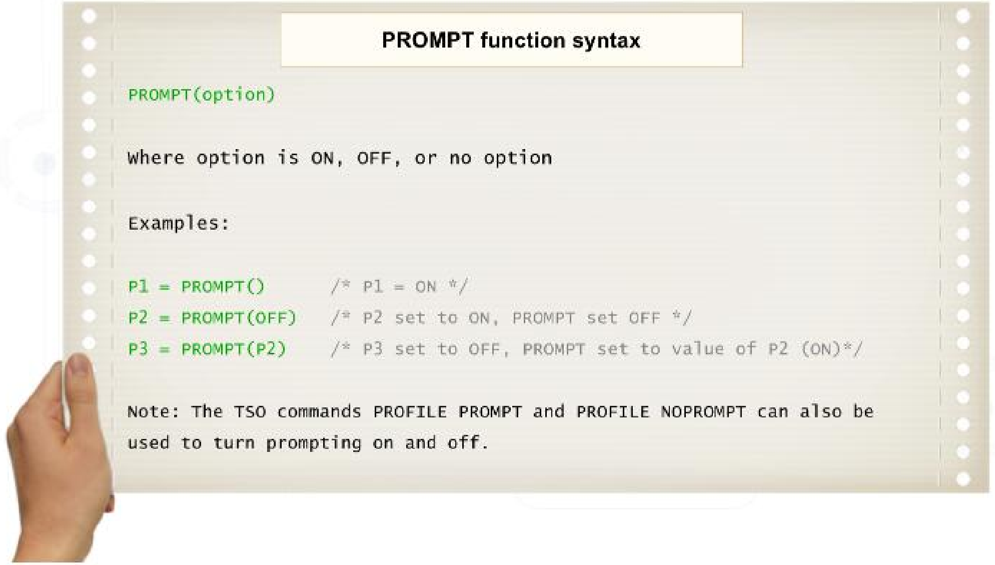


MSG function syntax

```
MSG(option)

Where option is ON, OFF, or no option

Examples:

M1 = MSG()       /* M1 = ON */
M2 = MSG(OFF)    /* M2 set to ON, MSG set OFF */
M3 = MSG(M2)     /* M3 set to OFF, MSG set to value of M2 (ON)*/
```

The MSG function enables the display of TSO/E informational messages from commands or functions to be switched on or off. This function can stop unwanted messages from TSO being displayed on the terminal. Not all messages are prevented from being displayed; it depends on the type of message and where it comes from.

No matter which option is used, the MSG function always returns the current value of the MSG facility before the function is performed. Using a variable when first switching MSG to OFF enables the program to return MSG to its original setting before exiting.



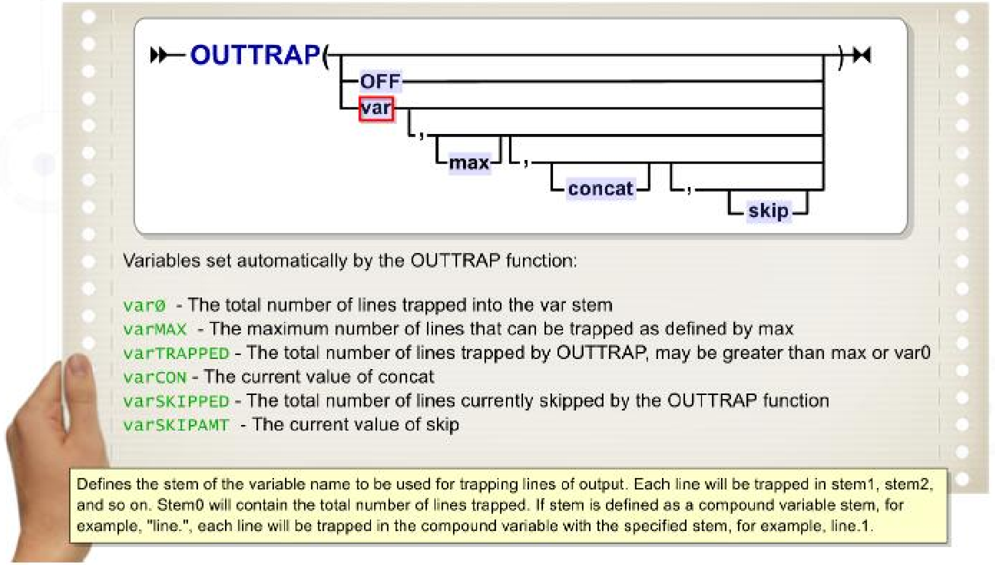
PROMPT function syntax

```
PROMPT(option)

Where option is ON, OFF, or no option

Examples:

P1 = PROMPT()       /* P1 = ON */
P2 = PROMPT(OFF)    /* P2 set to ON, PROMPT set OFF */
P3 = PROMPT(P2)     /* P3 set to OFF, PROMPT set to value of P2 (ON)*/

Note: The TSO commands PROFILE PROMPT and PROFILE NOPROMPT can also be
used to turn prompting on and off.
```

The PROMPT() function enables the TSO command prompting facility to be switched on or off. When prompting is on and an incomplete TSO command is entered, the system prompts the user for the missing parameters.

If this is inconvenient while executing a REXX, it may be preferable to allow the command to fail and check the return code to execute an error or recovery routine. When ON, prompts can be replied to by using data in the external data queue or stack.

The format of the PROMPT function is `X1 = PROMPT(option)` where option is ON, OFF, or no option; `X1` will be set to the current value of PROMPT and the prompting facility will be set to whichever option is specified.



Variables set automatically by the OUTTRAP function:

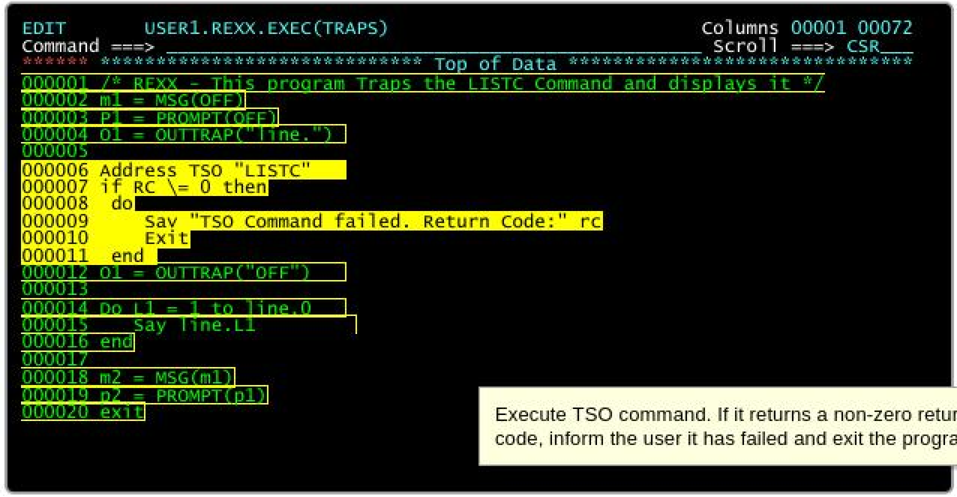
`var0` - The total number of lines trapped into the var stem
`varMAX` - The maximum number of lines that can be trapped as defined by max
`varTRAPPED` - The total number of lines trapped by OUTTRAP, may be greater than max or var0
`varCON` - The current value of concat
`varSKIPPED` - The total number of lines currently skipped by the OUTTRAP function
`varSKIPAMT` - The current value of skip

Defines the stem of the variable name to be used for trapping lines of output. Each line will be trapped in stem1, stem2, and so on. Stem0 will contain the total number of lines trapped. If stem is defined as a compound variable stem, for example, "line.", each line will be trapped in the compound variable with the specified stem, for example, line.1.

The OUTTRAP function traps messages from commands or programs that would normally be displayed on the terminal. For example, a user would normally list all the data sets starting with their TSO userid by entering LISTC on their TSO terminal. With the OUTTRAP function, each line produced by the command can be trapped into a stem variable defined by the function.

After executing the required command, OUTTRAP should be set to OFF to ensure no future commands overwrite the trapped lines. The value returned by the OUTTRAP function is a return code indicating whether the function call successfully completed.

Mouse-over the code for a description of each clause.

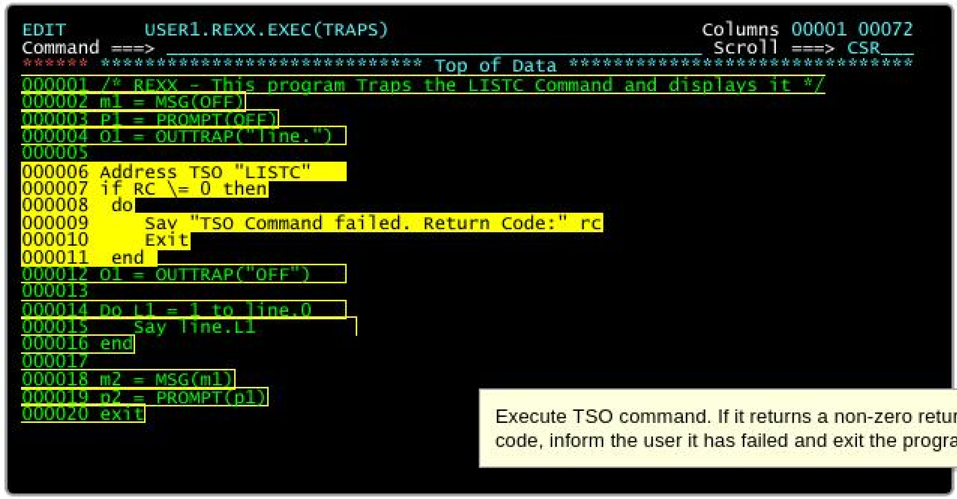


```
EDIT       USER1.REXX.EXEC(TRAPS)                            Columns 00001 00072
Command ===> ____________________________________________  Scroll ===> CSR___
****** ***************************** Top of Data ******************************
/* REXX - This program Traps the LISTC Command and displays it */
m1 = MSG(OFF)
P1 = PROMPT(OFF)
o1 = OUTTRAP("line.")

Address TSO "LISTC"
if RC \= 0 then
 do
    Say "TSO Command failed. Return Code:" rc
    Exit
 end
o1 = OUTTRAP("OFF")

Do L1 = 1 to line.0
    Say line.L1
end

m2 = MSG(m1)
p2 = PROMPT(p1)
exit
```

Execute TSO command. If it returns a non-zero return code, inform the user it has failed and exit the program.

This is an example of the TSO/E message trapping and controlling functions.

Mouse-over the code for a description of each clause.

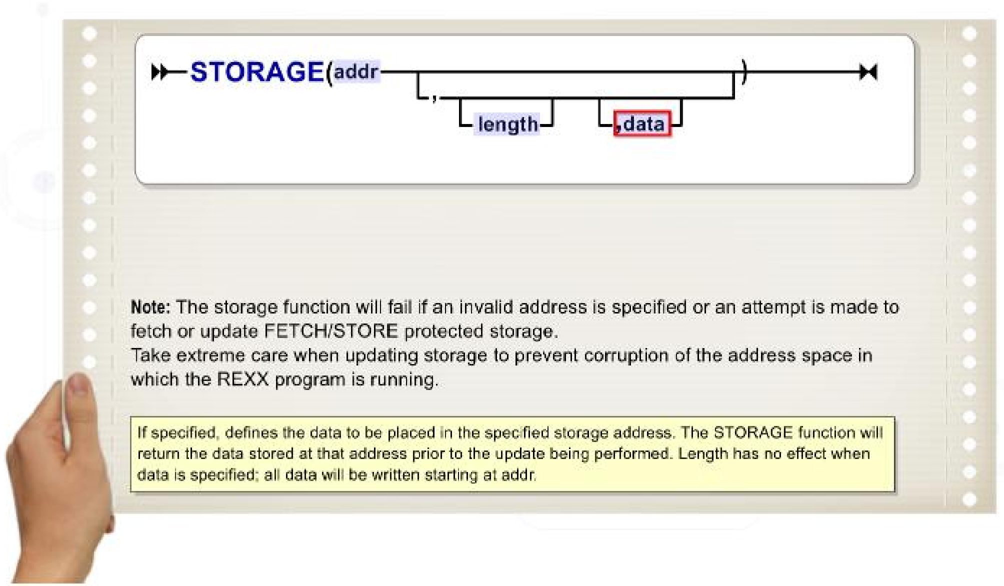


```
EDIT       USER1.REXX.EXEC(TRAPS)                               Columns 00001 00072
Command ===> ___________________________________________________ Scroll ===> CSR___
****** ************************** Top of Data ***************************
/* REXX - This program Traps the LISTC Command and displays it */
m1 = MSG(OFF)
P1 = PROMPT(OFF)
O1 = OUTTRAP("line.")

Address TSO "LISTC"
if RC \= 0 then
 do
    Say "TSO Command failed. Return Code:" rc
    Exit
 end
O1 = OUTTRAP("OFF")

Do L1 = 1 to line.0
   Say line.L1
end

m2 = MSG(m1)
p2 = PROMPT(p1)
exit
```

Execute TSO command. If it returns a non-zero return code, inform the user it has failed and exit the program.

This is an example of the TSO/E message trapping and controlling functions.

Mouse-over the code for a description of each clause.



Note: The storage function will fail if an invalid address is specified or an attempt is made to fetch or update FETCH/STORE protected storage.
Take extreme care when updating storage to prevent corruption of the address space in which the REXX program is running.

If specified, defines the data to be placed in the specified storage address. The STORAGE function will return the data stored at that address prior to the update being performed. Length has no effect when data is specified; all data will be written starting at addr.

The STORAGE function enables the user to interrogate and, in some instances, change the storage within the user's address space. This is mainly a systems programmer function that is rarely used in REXX coding.

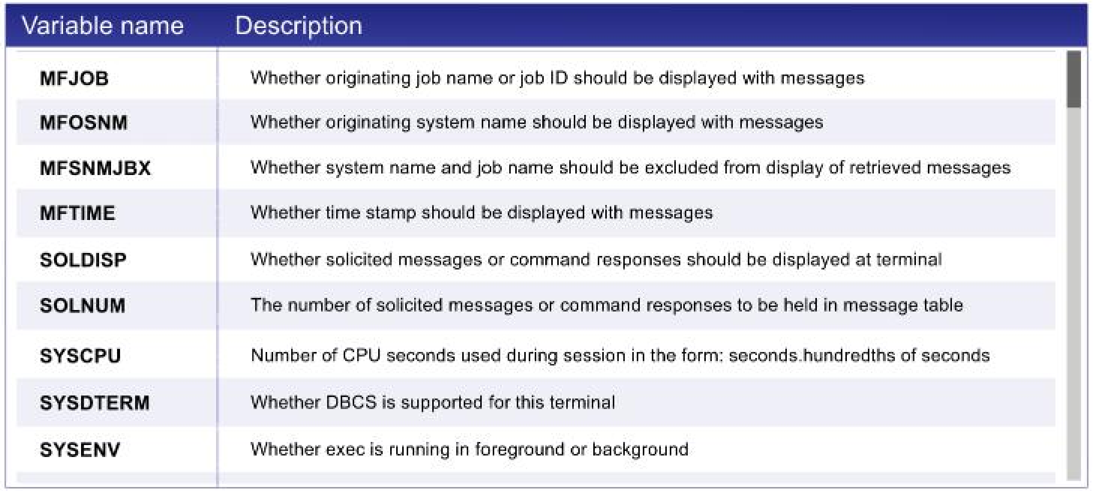
Mouse-over the code for a description of each clause.

Variable name	Description
SYSAPPCLU	The APPC/MVS logical unit (LU) name
SYSDFP	The level of MVS/Data Facility Product (MVS/DFP)
SYSMVS	The level of the base control program (BCP) component of z/OS
SYMDEF	Symbolic variables of your MVS system
SYSOPSYS	The z/OS name, version, release, modification level, and FMID
SYSSECLAB	The security label (SECLABEL) name of the TSO/E session
SYSSMFID	Identification of the system on which System Management Facilities (SMF) is active
SYSSMS	Indicator of whether DFSMS/MVS is available to your REXX exec
SYSCLONE	MVS system symbol representing its system name
SYSPLEX	The MVS sysplex name as found in the COUPLExx or LOADxx member of SYS1.PARMLIB
SYSNAME	The name of the system that your REXX exec is running on, as specified in the SYSNAME statement in SYS1.PARMLIB member IEASYSxx

The MVSVAR function enables a REXX program to interrogate several system variables in the OS/390 and z/OS environments, which describe MVS, TSO/E, and the current session environments. These include such data as the symbolic name of the MVS system. The format of the MVSVAR function is `X1 = MVSVAR(var)` where `var` is one of the variable names shown in the table displayed here.

When using the SYMDEF parameter, two values are passed to the MVSVAR function. The second refers to the symbolic name as defined in the COUPLExx, LOADxx, or IEASYMxx member of SYS1.PARMLIB.

For example, if the IEASYMxx member of SYS1.PARMLIB defined the variable &SYSNAME as 'SYSA', then:

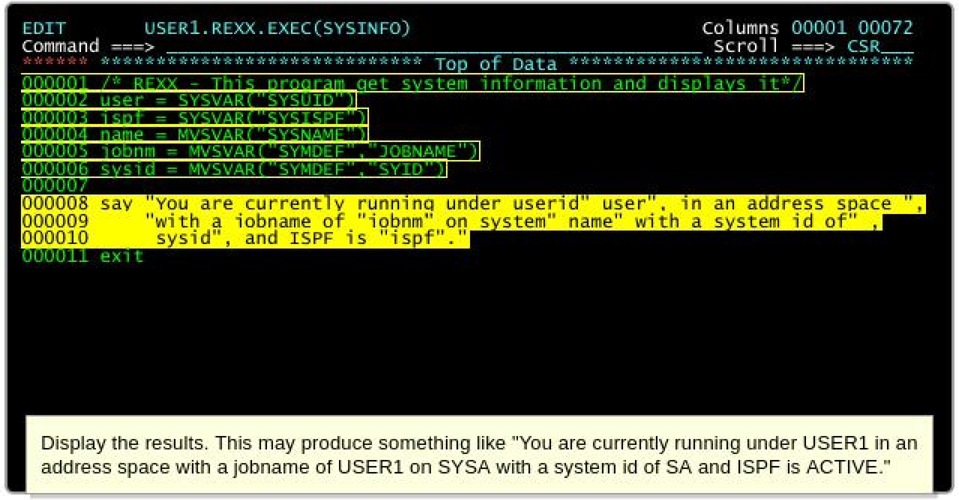


Variable name	Description
MFJOB	Whether originating job name or job ID should be displayed with messages
MFOSNM	Whether originating system name should be displayed with messages
MFSNMJBX	Whether system name and job name should be excluded from display of retrieved messages
MFTIME	Whether time stamp should be displayed with messages
SOLDISP	Whether solicited messages or command responses should be displayed at terminal
SOLNUM	The number of solicited messages or command responses to be held in message table
SYSCPU	Number of CPU seconds used during session in the form: seconds.hundredths of seconds
SYSDTERM	Whether DBCS is supported for this terminal
SYSENV	Whether exec is running in foreground or background

The SYSVAR function is similar to the MVSVAR function. It returns the value of system variables that are available within the TSO/E session, which enables the user to determine specific information about the TSO/E environment.

The format of the SYSVAR function is `X1 = SYSVAR(var)` where `var` is one of the variable names shown in the table displayed here. The variable name should be enclosed in quotes.

Scroll through the list of argument values and their descriptions. Refer to the TSO/E REXX Reference for more information.



```
EDIT       USER1.REXX.EXEC(SYSINFO)                        Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
/* REXX - This program get system information and displays it*/
user = SYSVAR("SYSUID")
ispf = SYSVAR("SYSISPF")
name = MVSVAR("SYSNAME")
jobnm = MVSVAR("SYMDEF","JOBNAME")
sysid = MVSVAR("SYMDEF","SYID")

say "You are currently running under userid" user", in an address space ",
    "with a jobname of "jobnm" on system" name" with a system id of" ,
     sysid", and ISPF is "ispf"."
exit
```

Display the results. This may produce something like "You are currently running under USER1 in an address space with a jobname of USER1 on SYSA with a system id of SA and ISPF is ACTIVE."

Mouse-over this example of the TSO/E system information functions for a description of each clause.