**REXX Notes**

**sites**

**Tutorials**

**Abends**

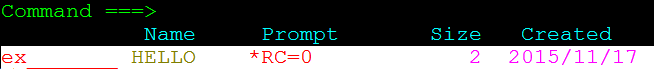
**Utilities –**

**Example Programs**

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

**Imp Points**

* **ABBREVATIONS**
* Supply EX before rexx program member in PDS to run the REXX pgm.



* Comments always start /\* and end \*/.
* Please note that starting letters of all rexx programs must be /\* REXX \*/
* If you want two instructions on one line you will need to put a semi-colon between them.
* Line continuation character is **Comma (,) for easytrive.**

**It is + for IDCAMS.**

**REXX and sort card it is ,**

* The maximum length of a literal is platform specific but it will be at least 100 bytes and normally considerably more.
* **SAY "To print without" || "gaps use these."** This use of Abuttal prints without any spaces
* Simple variable names can contain these chars.

A - Z, a - z, 0 - 9, !, ?, #, $, @ & \_

0 - 9 can not be used for the first character.

All lowercase chars. are converted to uppercase.

Variables are initialized with their name!

**EG:-**

SAY “New\_var=” new\_var

ANS. NEW\_var= new\_var

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question and Answers**

* REXX was originally provided by IBM as a component of the mainframe
* REXX is a high-level language designed to support personal programming,

operating system command files, macros, and prototyping.

**Q. How to access data on JES SPOOl?**

ANS. Output held on JES2 or JES 3 can be captured by outtrapping the output of following command.

“OUTPUT jobname PRINT(\*) KEEP”

Use command HELP OUTPUT for information

**->SAY** instruction

template: **SAY** expression

EG:

1. **SAY** "print this line"

2. **SAY** "Contents of A=" A

**-> Rules**

There really are very few rules when it comes to coding REXX. However it is important to understand the few that there are.

1. The system presumes that each line is a new instruction unless you state you are continuing on another line by the use of a comma (we will see this in a minute).
2. If you want two instructions on one line you will need to put a semi-colon between them.
3. You can indent your instructions as much as you like; this is very useful when you are coding loops.
4. You can also have as many blank lines as you like. These can be used to section your code and make it far more readable.

**Concatenation**A very important concept it the automatic concatenation that applies within REXX. This is not only used for printing but in lots of other situations. The basic idea is that if you put two things (TOKENS) next to each other (in this example two literal strings), then they will be connected with a space. Note that they are always joined by one space regardless of how many spaces there are between the literal strings.   
**Abuttal**  
If you do not wish to have a space between the two tokens you must code two vertical lines (OR symbol). This system can apply to numbers as well, we haven't looked at these yet but what do you think A would equal here?

**A = 5 || 7**

* **Variables**

1. **Naming Convention-** If your variable name contains lower case characters then these are automatically converted into capitals. See here how the variable TOTAL\_a is used in one line and on the next it is referred to a TOTAL\_A. Doing this for real would be considered a very sloppy habit but it’s quite legal (like a lot of sloppy habits).
2. **Initialized Value**

This is something worth remembering, in REXX you can use a variable even if you have not given it a value. In the example here we have only used NEW\_var on a SAY statement. So what does it print? The answer is its own name (in capitals of course).

1. **Data Type**

Another rather weird thing about REXX is that you do not have to tell it whether your variable is going to contain a string of characters or a number - it just doesn't care!

* **Arithmetic Operators**

/\* REXX program ARITHMETIC operators. \*/

SAY "ADD: 5 + 2 =" 5 + 2 ; SAY "SUBTRACT:2 - 5 =" 2 - 5

SAY "MULTIPLY:5 \* -2 =" 5 \* -2 ; SAY "DIVIDE: 5 / +2 =" 5 / +2

SAY "INT. DIV.5 % 2 =" 5 % 2 ; SAY "REMAINER:5 // 2 =" 5 // 2

SAY "POWER: 5 \*\* 2 =" 5 \*\* 2

SAY "ABUTTAL:5 || 2'00' =" 5 || 2'00'

SAY "USE BRACKETS FOR PRIORITY:" 7 || ((2 + 5) \* 5)/3

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

ADD: 5 + 2 = 7

SUBTRACT:2 - 5 = -3

MULTIPLY:5 \* -2 = -10

DIVIDE: 5 / +2 = 2.5

INT. DIV.5 % 2 = 2

REMAINER:5 // 2 = 1

POWER: 5 \*\* 2 = 25

ABUTTAL:5|| 2'00' = 5200

USE BRACKETS FOR PRIORITY: 711.6666667

Operators

Here you can see the different mathematical operators available. It all looks pretty straight forward until you come down to ABUTTAL - what is this doing here you may say? Well I said you could use concatenations for more than just pretty SAY statements! Two types of abuttal are used here, one using the '||' symbol and the other is achieved by typing the CONSTANT next to the literal string. We haven't seen this method of abuttal before. Note the last number in the example 711.666667, the precision here means we get a total of 9 digits. This is an environmental option which we can change with the NUMERIC keyword we'll see this later.

Operator Priorities & Brackets

These can get silly, I say that because we could spend a lot of time learning them and then writing very fancy code which works and is very clever. Clever but unreadable when two months later when you can't remember which operator has priority over its neighbour. The solution is simple use brackets even when they may not be needed! If you want to be an anorak here are the priorities, when two are the same, the priority is from left to right.

8] + - (as a prefix) 5] + - (not as a prefix)

7] \*\* 4] Abuttal.

Expresssions

What is an expression?

Simply its several bits of code which will be used to work out one thing, that 'thing' may be a literal string or a number. Let's take an assignment instruction for example

**OUTLINE = 'The Total is =' L \* 2 + W \* 2 + ExtraBit**

Everything here except the OUTLINE = is one expression. It contains constants, literal strings, arithmetic operators and variables but all the values will be shrunk down to one literal string that in this case will be put into the variable called OUTLINE. We could however print it by coding SAY at the beginning rather than OUTLINE =. Later we will look at functions and the arguments, which are their inputs. Normally we have just a single variable as an argument but in fact it will take a very large and complex expression.

* **The PULL instruction**

/\* REXX program to show the PULL instruction \*/

SAY "Program to multiple 2 numbers"

SAY "Please enter your first number"

PULL Num1

SAY "Thank you, now your second number"

PULL Num2

SAY "Well done: Now" Num1 "times" Num2 "=" Num1 \* Num2

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

Program to multiple 2 numbers

Please enter your first number

12

Thank you, now your second number

14

Well done: Now 12 times 14 = 168

The PULL Instruction is a funny little baby. What it does will depend on where you are executing your program. When running it online it will pause the execution of the routine and wait for you to type something in. When executing the REXX in JCL this instruction will try and get a line from the input queue - more about that later. For now we will worry about running it online. In the example above I typed in the numbers 12 and 14 in 'answer' to the two PULL statements. Its a pretty obvious thing to say but if you are going to use PULL like this then have a good SAY instruction just before it so you will know what to type in!

There is far more to PULL than meets the eye, you can not only accept input with it but also split it up, convert it to upper case, its very clever.

**Note: some time if we come near end of page we might need to press enter before giving value for PULL variable.**

**Chapter 19: REXX Support**

This chapter describes how REXX is supported in Mainframe Express.

**Introduction**

REXX is a high-level language with extensive facilities for manipulating text. REXX programs can also include host system commands and so can be used in place of TSO CLISTs.

For more information about the REXX language and how it is used, see the [IBM OS/390 TSO/E REXX Reference](http://www.microfocus.com/docs/links.asp?mfe=os390ref" \t "MFWin) and [IBM OS/390 TSO/E REXX User's Guide](http://www.microfocus.com/docs/links.asp?mfe=os390ug).

A REXX program is called an exec. Before running an exec, you need to add the exec to your project, or download it from the mainframe, in the usual way. All REXX execs should start with a comment containing the character string REXX so that they can be distinguished from TSO CLISTs. For example:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* REXX \*\*\*\*\*\*\*\*\*\*\*\*\*\*/

On the PC, REXX execs should have the file extension **.rex**. By default, they are placed in the REXX source library in the Workgroup location.

Optionally, you can define an exec as a member of a partitioned data set (PDS). The DD names SYSEXEC and SYSPROC are normally used for PDSs containing REXX execs. More than one PDS can be allocated to each of these DD names. A SYSEXEC PDS can contain only execs. A SYSPROC PDS can contain CLISTs and execs.

When you run the exec, if you do not specify the location of the exec explicitly, Mainframe Express uses the following search order to locate it:

1. SYSEXEC data sets
2. REXX source library in the Workgroup location
3. SYSPROC data sets
4. CLIST source library in the Workgroup location

Mainframe Express interprets an exec as it is run. There are no preceding compilation and linking stages.

Mainframe Express supports the use of REXX execs in the following host environments:

* [*TSO interactive (foreground) environment*](https://supportline.microfocus.com/documentation/books/mx31/ugrexx.htm#u002). This is primarily for execs that need interaction with the user.
* [*TSO batch (background) environment*](https://supportline.microfocus.com/documentation/books/mx31/ugrexx.htm#u003). This is for execs that do not need interaction with the user but still need access to TSO services.
* [*JCL (non-TSO) environment*](https://supportline.microfocus.com/documentation/books/mx31/ugrexx.htm#u004). This can be used for execs that do not need interaction with the user and do not need access to TSO services.

**TSO Interactive Environment**

In the TSO interactive environment, you can run a REXX exec in the same way as a CLIST. You can specify the exec that you want to run:

* On the **TSO** tab of the **Start Debugging** dialog box.

or

* On an EXEC statement at the TSO READY prompt.

For example, if the exec is named **tsotest** and is a member of a private PDS named **userid.rexx.exec**, you can specify one of the following forms:

* ex 'userid.rexx.exec(tsotest)' exec
* ex rexx.exec(tsotest) exec
* ex rexx(tsotest) exec

The final exec keyword identifies it as a REXX exec. You can omit it if the first line of the exec is a comment containing the character string REXX.

If the exec is a member of a PDS allocated to SYSEXEC or SYSPROC, or if SYSEXEC and SYSPROC are not defined but the REXX exec is included in the project, you can simply specify one of the following:

* tsotest
* %tsotest

**TSO Batch Environment**

You can use the IKJEFT01 utility to set up a TSO batch environment and run a REXX exec in the same way as a CLIST.

For example, the following JCL jobstream runs the REXX exec named **tsotest** in the PDS named **userid.rexx.exec**.

//jobname JOB ............

//\*

//jobstep EXEC PGM=IKJEFT01,.........

//SYSEXEC DD DSN=USERID.REXX.EXEC,DISP=SHR

//SYSTSPRT DD SYSOUT=A

//SYSTSIN DD \*

%TSOTEST

/\*

//

You can run the JCL jobstream from the **Debug** menu in the usual way.

**JCL Environment**

In the JCL environment, you can use the IRXJCL utility to run a REXX exec.

For example, the following JCL jobstream runs the REXX exec named **jcltest** in the PDS named **userid.rexx.exec** that has been allocated to SYSEXEC.

//jobname JOB ............

//\*

//jobstep EXEC PGM=IRXJCL,PARM='JCLTEST'

//SYSEXEC DD DSN=USERID.REXX.EXEC,DISP=SHR

//SYSTSPRT DD SYSOUT=A

//SYSTSIN DD \*

input to exec

/\*

//

**Passing Parameters**

You can pass parameters to a REXX exec in the usual way, for example:

* rexx(add) 42 21 exec
* add 42 21
* %ADD 42 21
* //jobstep  EXEC PGM=IRXJCL,PARM='ADD 42 21'

In the exec named **add**, you can use an ARG instruction to receive the two parameters, 42 and 21.

**Including Host Commands**

You can include a host command in a REXX exec by enclosing it in quotation marks, for example:

"EXECIO \* DISKR MYINDD (FINIS STEM MYVAR"

The REXX language processor passes the command and any other source code that it does not recognize to the host command environment for execution.

Mainframe Express emulates the TSO and MVS host command environments..It selects the appropriate environment by default on entry to a REXX exec. TSO is selected for the TSO interactive and batch environments and MVS is selected for the JCL environment. If you use a host command that is not recognized by the current environment, the return-code variable RC is set to -3.

You can use the ADDRESS built-in function to find the name of the current environment and the ADDRESS instruction to change it.

**Debugging an Exec**

You can use the REXX debugging facility to help you debug an exec. It lets you interact with the exec while it is running and monitor the values of variables.

The TRACE instruction lets you start and control the debugging facility from within an exec.

The EXECUTIL TS command entered at the TSO READY prompt starts the debugging facility for the next REXX exec that you invoke.

**Specifying Record Length for Zero-length Line Sequential Files**

The MFREXX\_LSEQ\_RECSIZE\_MAX environment variable enables you to specify the record length for the REXX EXECIO command when reading or writing a line sequential file that has been cataloged with a record length of zero.

The format of the environment variable is:

SET MFREXX\_LSEQ\_RECSIZE\_MAX=*rec-length*

where *rec-length* is the record length to use. The default is 255, which is the minimum value you can use. Specifying an invalid value, or a value less than 255, results in the default value being used.

**Specifying Alternative Characters for the OR and NOT Operators**

If you need to use characters other than the defaults for REXX's OR and NOT operators, Mainframe Express enables you to do this using two environment variables: MFREXX\_OR and MFREXX\_NOT. You must set these environment variables before you start Mainframe Express for them to take effect.

You can specify one or more characters for each operator, and each character you specify is used in addition to the default operator; that is "|" for OR and "¬" for NOT. When specifying additional characters, you must ensure that each additional character is defined in single or double quotes, or in hexadecimal. When specifying more than one character, separate each one with a space or comma. Any error in the specification of the additional characters, including the specification of alphabetic or numeric characters, results in the environment variable being ignored.

**Examples:**

SET MFREXX\_NOT="^"

SET MFREXX\_NOT=0x5e "!"

SET MFREXX\_NOT=X'5e' '!' "~"

**Restrictions**

The following mainframe REXX facilities are not supported by Mainframe Express:

* Sequence numbers in the last eight bytes of fixed-length source records are recognized but no checking is done on the order of these numbers. Sequence numbers in variable-length records are not recognized.
* The TRACE S instruction is syntactically allowed but does not trace the remaining clauses in the exec. The exec is fully syntax checked when it is first loaded.
* Numeric options entered during interactive debugging are not supported.
* Console sessions are not supported; the TSO external function GETMSG is syntax checked and the return-code variable RC is set to 12.
* The TSO external function LISTDSI only sets the following variables:
  + SYSBLKSIZE
  + SYSDSNAME
  + SYSDSORG
  + SYSDSSMS
  + SYSLRECL
  + SYSMEMBERS
  + SYSPRIMARY
  + SYSRECFM
  + SYSSECONDS
  + SYSUNITS
  + SYSVOLUME

All the other variables normally set by LISTDSI are set to a NULL string.

* The TSO external function MVSVAR only sets the following variables:
  + SYSDFP
  + SYSMVS
  + SYSNAME
  + SYSOPSYS
  + SYSSMS

All the other variables normally set by MVSVAR are set to a NULL string.

* The TSO external function STORAGE always returns a NULL string, but all parameters are syntax checked.
* The TSO external function SYSVAR only sets the following variables:
  + MFJOB
  + MFOSNM
  + MFSNMJBX
  + MFTIME
  + SOLDISP
  + SOLNUM
  + SYSCPU
  + SYSDTERM
  + SYSENV
  + SYSISPF
  + SYSKTERM
  + SYSLTERM
  + SYSNEST
  + SYSNODE
  + SYSPLANG
  + SYSPREF
  + SYSRACF
  + SYSSLANG
  + SYSSRV
  + SYSTSOE
  + SYSUID
  + SYSWTERM
  + UNSDISP
  + UNSNUM

All the other variables normally set by SYSVAR are set to a NULL string.

* The TSO REXX command EXECUTIL only supports the following options, but all parameters are syntax checked:
  + HI
  + HT
  + RT
  + TE
  + TS

All other options are treated as NOPs.

* CICS REXX is not supported.
* Compiled REXX execs are not supported.
* APPCMVS, ATTACH, ATTACHMVS, ATTACHPGM, CONSOLE, CPICOMM, ISPEXEC, ISREDIT, LINK, LINKMVS, LINKPRM, and LU62 host command environments are not supported:
* User-defined host command environments are not supported.
* TSO external functions, external subroutines, and function packages are not supported.
* The following REXX programming services are not supported:
  + IRXEXCOM - variable access routine
  + IRXEXEC - exec processing routine
  + IRXHLT - halt condition routine
  + IRXIC - trace and execution control routine
  + IRXINIT - initialization routine
  + IRXLIN - LINESIZE function routine
  + IRXRLT - get result routine
  + IRXSAY - SAY instruction routine
  + IRXSUBCM - maintain entries in the host command environment table
  + IRXTERM - termination routine
  + IRXTXT - text retrieval routine
* Replaceable routines and exits are not supported.
* DBCS is not supported.

## Question. How to execute REXX Procedure with shortcut or using TSO

## Answer

**Allocate and activate your user EXEC library when you start a session**

TSO ALLOCATE DDNAME(SYSUEXEC) DSN(TEST.EXEC) SHR REUSE  
TSO ALTLIB ACTIVATE USER(EXEC)

**Enter % followed by the name of the REXX procedure to be run**

TSO %MKTGTEST

## Question. How to execute CLIST with shortcut or using TSO

**Answer**

**Allocate and activate your user CLIST library when you start a session**

TSO ALLOCATE DDNAME(SYSUEXEC) DSN(TEST.CLIST) SHR REUSE  
TSO ALTLIB ACTIVATE USER(EXEC)

**Enter % followed by the name of the CLIST to be run**

TSO %MKTGTEST

**Question. How can you capture data or screens that are thrown from CLIST invocation?**

**ANSWER**

OUTTRAP returns the name of the variable in which trapped output is stored, or if trapping is not in effect, OUTTRAP returns the word off.

x = OUTTRAP('VAR.')

"%REXX2"

y = OUTTRAP('OFF')

EXIT

/\* REXX2 command \*/

SAY "This is output from the REXX2 command " /\* This will be trapped \*/

RETURN

**Question How to invoke CLIST or TSO commands from REXX**

**Answer**

I general, if you need to prefix your command with "TSO", then you should be able to invoke it from rexx with something like this:

|  |
| --- |
| Code: |
| Address TSO "%yourclst" |

If your command is not prefixed with "TSO" and it needs to run from within the ISPF editor, then you can invoke it from rexx with something like this:

|  |
| --- |
| Code: |
| Address ISREDIT "MACRO" "%yourclst" |