



Reading and Writing Data Using REXX EXECIO on OS/390 and z/OS

By Lionel Dyck

This article offers tips on reading and writing data using the REXX programming language and the EXECIO command and concludes with some useful resources on REXX.

ONCE you learn REXX, one of the things that you will find very useful is the ability to read data from a dataset and to write data to a dataset. Fortunately for the REXX user, this is as easy as doing anything else in REXX.

This article does assume that the reader has a basic understanding of TSO commands such as Allocate and Free, of ISPF, as you will be using the ISPF Editor to code your REXX programs (also known as REXX Execs), and of basic REXX coding and syntax.

The REXX programming language was developed in the VM environment (if you don't remember what VM is I suspect that you will in the near future if your installation is considering running Linux on your S/390 or zSeries processor). VM, or Virtual Machine, is an operating system that IBM developed 30 years ago to provide each user with their own virtual personal computer. Today's VM is known as z/VM.

The reason for that short history lesson is because the command syntax for reading and writing data using REXX is based on a VM command called EXECIO, or Execute I/O. This syntax is different from the syntax you are used to with the CLIST programming language and with the majority of TSO commands. This doesn't make it worse, or better, just different and that difference you will need to remember as it could cause you coding problems when you code the EXECIO command. Be aware that the EXECIO method of reading and writing is not part of the REXX standard, so you will not find it in other implementations (e.g. Regina REXX).

READING DATA FROM A DATASET

Let's start by reading data from a dataset. To do this you will need to first allocate the dataset using the TSO Allocate command. The command syntax in REXX for this is in FIGURE 1.

The ddname in the ALLOC command is the same as the ddname on the DD statement in JCL. The shr is the same as the DISP=, or disposition, on the DD statement. The ds is the same as the DSN= on the DD statement. The dataset name must be a sequential dataset or a member

FIGURE 1: ALLOCATING A DATASET

```
"Alloc F(ddname) shr ds(dataset-name)"
```

FIGURE 2: READING A DATASET

```
"Execio * Diskr ddname (Finis Stem in.)"
```

FIGURE 3: READING A DATASET WITH THE DDNAME AS A VARIABLE

```
ddname = "input"  
"Execio * Diskr" ddname "(Finis Stem in.)"
```

FIGURE 4: CLOSING A READ DATASET

```
"Execio 0 Diskr ddname (Finis)"
```

of a partitioned dataset, as REXX can only read sequential data with the EXECIO command.

The EXECIO command in REXX is very powerful, however for the purpose of this introductory article, I will focus on the usage that I have come to prefer over the years. See FIGURE 2 for an example of the syntax for reading the dataset.

In this example, the command is EXECIO. It is enclosed within quotes along with all of the parameters used. If any of the parameters are variables, then those variables would be outside the quotes—see FIGURE 3 for an example where the ddname is a variable.

You should have noticed by now that REXX is not case sensitive. This is a good thing as it makes reading REXX code much easier, and important parts of the code can be highlighted by using capital letters.

In FIGURE 2 and 3 the * is used to indicate to EXECIO to read all of the records in the input dataset. If you only needed to read a few records

then the * could be replaced by that number. You would then have to repeat the EXECIO multiple times to read the entire dataset. The DISKR keyword is used to indicate that the dataset is to be read and not written.

The ddname is the same ddname used in the previous ALLOC command. This is because REXX needs to have the dataset already allocated before it can access any of the records in the dataset.

The left parenthesis, (, is an artifact of the VM heritage of the command and is used to indicate that additional options follow. In this case the Finis is a keyword used to tell EXECIO to close the dataset when it is finished reading all the records. You would not use this if reading only a few records at a time, then you would close the dataset using code similar to FIGURE 4 (on page 22). The keyword Stem indicates that the records are to be read into the stem variable that follows the Stem keyword. In this case the stem variable is in. and the period after the variable name indicates that it is a stem variable.

RELEASING THE DATASET ALLOCATION

At this point the dataset has been processed, and all the records are in the stem variable. As the dataset will no longer be used, the dataset allocation can be released using the Free command as shown in FIGURE 5. The keyword F or File references the ddname of the allocation. This is normal TSO syntax, where the keyword parameter is enclosed in parenthesis compared to the VM syntax of EXECIO, where the keyword parameter is just coded next as in the Stem keyword and the stem variable name.

At this point the dataset has been allocated, all the records have been read into the stem variable in., and the dataset allocation has been released. You are now ready to begin processing all of the records that were just read from the dataset.

PROCESSING THE RECORDS

The stem variable in.0 now contains a count of the number of records that were read in. This count is very helpful should you want to process each record individually using a DO loop.

After processing the records you will want to write out the modified records, a report of some kind, or just create a copy of the records into a new dataset. The steps to write out the records are very similar to the steps to read in the records. First the output dataset must be allocated, then the records written out, and finally the dataset allocation released.

FIGURE 5: FREEING A DATASET ALLOCATION

```
"Free F(ddname)"
```

FIGURE 6: WRITING TO A DATASET

```
"Execio * Diskw ddname (Finis Stem out.)"
```

FIGURE 7: THE COMPLETE PROGRAM

```
/* REXX exec to find and report all datasets */
arg input
"Alloc F(in) DS("input") shr"
"Execio * diskr in (finis stem in."
"Free F(in)"
j = 0
do i = 1 to in.0
  if pos("DSN=",in.i) = 0 then iterate
  parse value in.i with . "DSN="dsn .
  parse value dsn with dsn"," .
  j = j + 1
  out.j = dsn
end
"Alloc f(out) ds(*)"
"Execio * Diskw out (finis stem out."
"Free f(out)"
```

To allocate the output dataset, the ALLOC command is once again used. Assuming the output dataset has already been allocated and exists, then the command used in FIGURE 1 (on page 22) will work again. If you need to allocate a new dataset, then you will need to specify SPACE and DCB information. See the TSO Help information for the ALLOC command for more information on the available options, and there are many options you can use. If you can do it in a JCL DD statement then you can do it in an ALLOC command (with the exception of allocating a GDG dataset which ALLOC can not do.

WRITING THE RECORDS OUT

To write the records out, you will use the EXECIO command once again. The syntax will be similar to what you've already seen, with the exception that the DISKR will be replaced by DISKW, for Disk Write. See FIGURE 6 for an example of how to write all the records to an output dataset.

Useful REXX reference materials are:

The REXX Language Association on the Web at <http://www.rexxla.org>.

An excellent list of REXX Publications on the Web at <http://www2.hursley.ibm.com/rexx/rexxbook.htm> maintained by the originator of REXX, Mike Cowlshaw.

The IBM REXX Web site at <http://www2.hursley.ibm.com/rexx>.

The IBM REXX Product Web page at <http://www-4.ibm.com/software/ad/rexx>.

The IBM z/OS 1.2 TSO/E REXX Reference at <http://publibz.boulder.ibm.com/epubs/pdf/ikj4a310.pdf>.

The IBM z/OS 1.2 TSO/E REXX User's Guide at <http://publibz.boulder.ibm.com/epubs/pdf/ikj4c310.pdf>.

The IBM REXX Compiler Web site at <http://www-4.ibm.com/software/ad/rexx/rexxcompd.html>.

If you are new to REXX check out Jeff Glatt's site which claims you can "Learn REXX Programming in 56,479 steps" at <http://www.borg.com/~jglatt/rexx/scripts/language/language.htm>


And you can find a lot of REXX examples for OS/390 and z/OS at my Web site at <http://www.lbdsoftware.com>. Look on both the ISPF and the TCP/IP pages.

In this example along with using the DISKW keyword, the stem variable name is now out. to indicate that the output records have been processed from the in. stem and put into the out. stem.

When writing out a stem variable in this way, the writing will stop when the first null variable is reached. Since the stem variable is initialized with all stems null, this is not a problem, as records are added to the stem variable sequentially (e.g. 1, 2, 3, ... instead of 1,3,5,7). It is also good practice to place the number of records in the stem.0 variable, although EXECIO with DISKW does not use it.

See FIGURE 7 (on page 24), which demonstrates how to read a dataset, find all records with DSN= in them, and then generate a list of all datasets (DSN) in the input dataset. The output dataset in this example is a *, which references the active TSO user's console. This example uses the arg function to take the dataset name of the input dataset as a parameter passed by the user when the command is called. A DO loop is used to scan through all the records and the parse function is used to extract the dataset name from the DSN= fields on the scanned records.

CONCLUSION

This is another article written by the author introducing REXX coding examples, hints, and tips. Other articles by the author about REXX are "Introduction to OS/390 REXX" published in Technical Support in March 2001 and "How to Parse Command Line Parameters in REXX," published in the June 2002 issue of Technical Support. 



NaSPA member Lionel B. Dyck is a lead MVS systems programmer for a large HMO in California. He has been in systems programming since 1972 and has written numerous ISPF dialogs over the years. His current project is evaluating Linux on the S/390 and zSeries platform. Lionel is an active member of NaSPA and SHARE. He can be contacted via e-mail at Lionel.B.Dyck@kp.org.

Technical[®]
and Operating Environments

SUPPORT