TSO/E File Control

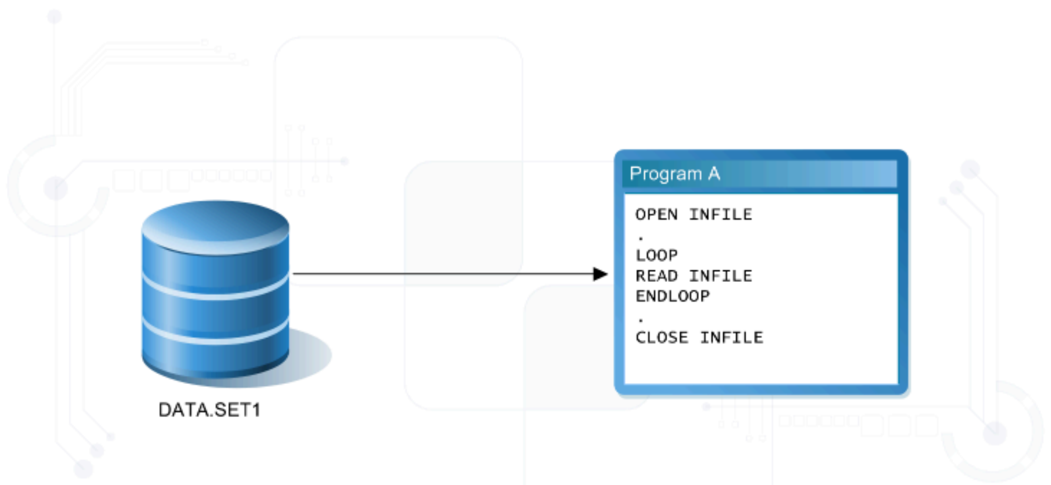
Objectives

TSO/E File Control

In this module, you will discover how the EXECIO TSO/E REXX command is used to read, write, and update data sets in the z/OS TSO/E environment.

After completing this module, you will be able to:

- Identify How to Read z/OS Data Sets By Using REXX
- Identify How to Write to z/OS Data Sets By Using REXX
- Identify How to Update z/OS Data Sets By Using REXX



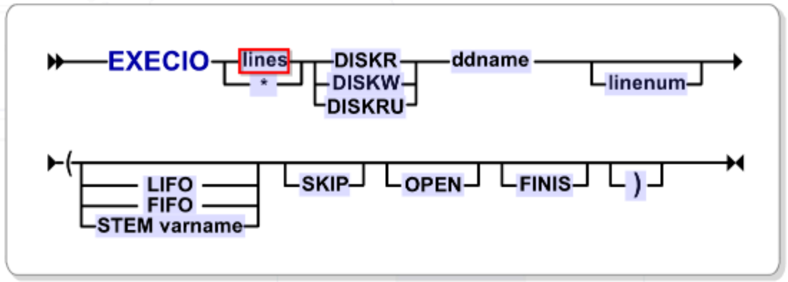
```
OPEN INFILE
.
LOOP
READ INFILE
ENDLOOP
.
CLOSE INFILE
```

When all required data set processing is complete, the data set is closed by the CLOSE macro or instruction.

Traditional languages like COBOL and PLI follow these three steps to process data sets after they have been allocated to the program:

1. Open the data set.
2. Process the records from the data set.
3. Close the data set.

These steps are separate instructions in the program and they must be performed in the correct order for the program to work.

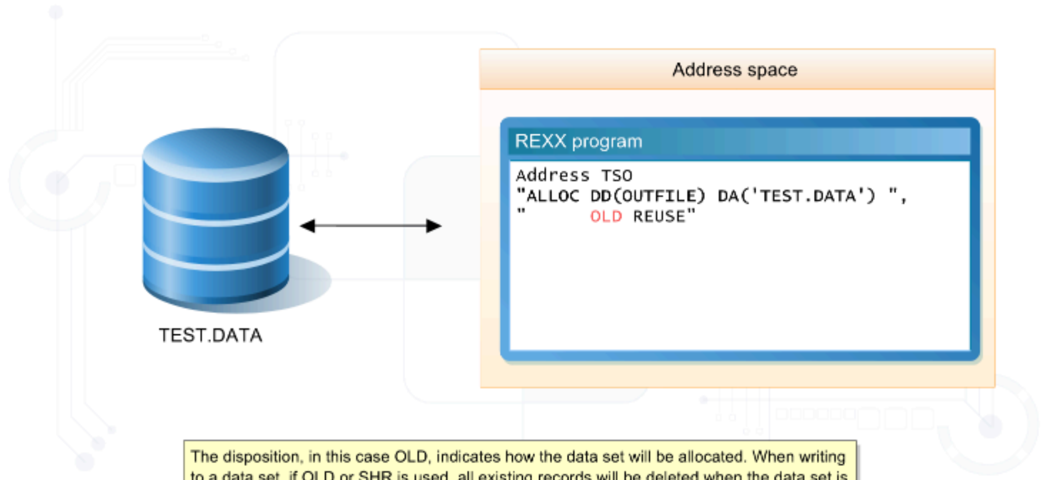


The number of lines to be read or written in this operation. Until the data set is closed, the system will "remember" which line is to be processed next by using a current row pointer (CRP).

The ENCIO command was used to perform all three stages of file processing on the original platform on which REXX was developed. A version of this command was ported to the MVS system platform under TSO/E so this is generally used for file processing in this environment when using REXX. The EXECIO command can only be used in REXX programs.

REXX programs can only directly process sequential data sets and individual members of partitioned data sets (PDS). Other data set organizations, such as VSAM, must be converted before REXX can process their records.

Mouse-over the syntax of the EXECIO command for a description of each parameter.



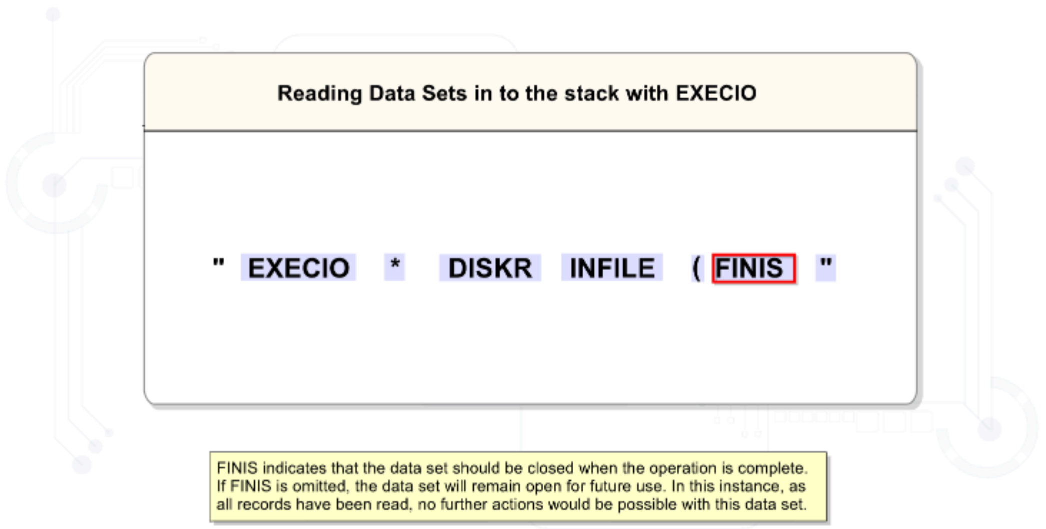
```
Address TSO
"ALLOC DD(OUTFILE) DA('TEST.DATA') ",
"       OLD REUSE"
```

The disposition, in this case OLD, indicates how the data set will be allocated. When writing to a data set, if OLD or SHR is used, all existing records will be deleted when the data set is opened. If MOD is specified, records will be added to the end of the data set. SHR is not recommended for output operations. When specifying NEW, other parameters are required.

Before using EXECIO, the required data set must be allocated to the address space that the REXX is running under. This can be done with a TSO ALLOC command when running under TSO/E, or with a JCL DD statement defined in the procedure or JOB that initiates the address space, that is, the TSO logon procedure or a batch JOB.

Standard dispositions of NEW, SHR, OLD, or MOD apply in both cases and should be carefully considered when reading or writing data sets. This is normally NEW or OLD for writing, SHR for reading, and MOD to enable records to be written to the end of a data set. After using the ALLOC command in a REXX program, it should be released back to the system by using the FREE command.

Click **Play** to see how data sets can be allocated for use by REXX programs.



Reading Data Sets in to the stack with EXECIO

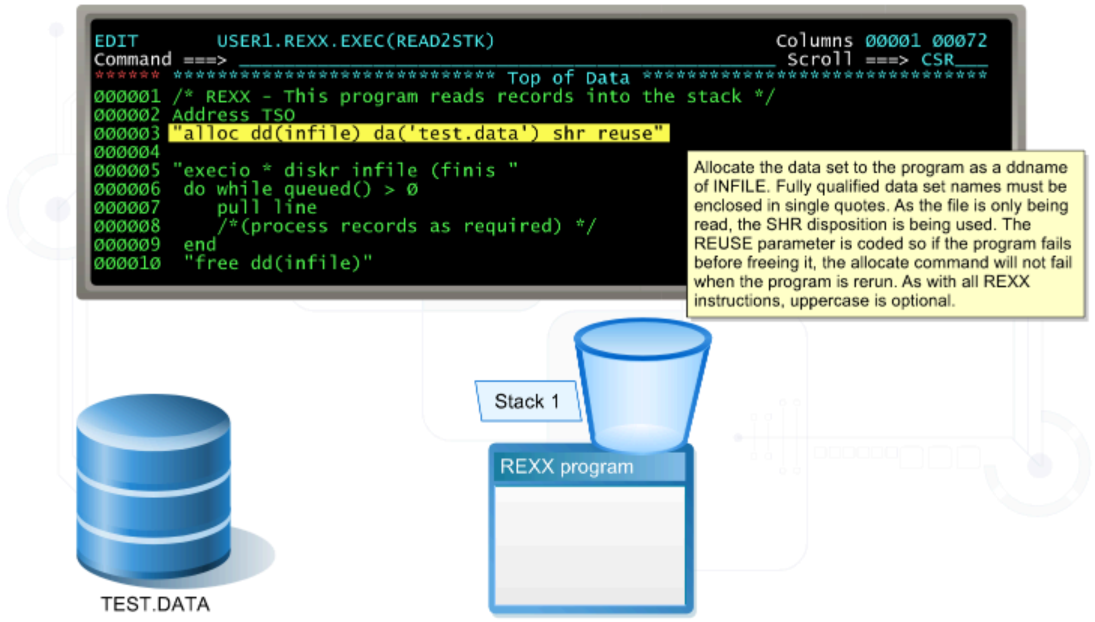
```
" EXECIO * DISKR INFILE ( FINIS "
```

FINIS indicates that the data set should be closed when the operation is complete. If FINIS is omitted, the data set will remain open for future use. In this instance, as all records have been read, no further actions would be possible with this data set.

After the data set has been allocated and the ddname is known, it can be read by the EXECIO command using the DISKR parameter.

The simplest form of the EXECIO command is shown above. This command will open the data set allocated to the ddname INFILE, read all the records from the data set, place them on to the stack, and then close the file.

Mouse-over the command for a description of each parameter.

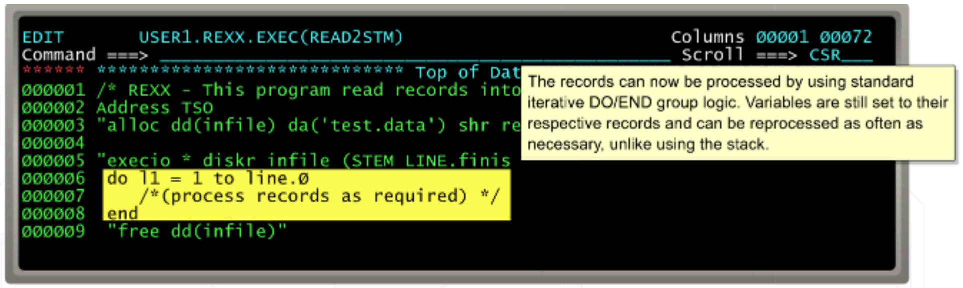


```
EDIT       USER1.REXX.EXEC(READ2STK)                          Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
/* REXX - This program reads records into the stack */
Address TSO
"alloc dd(infile) da('test.data') shr reuse"

"execio * diskr infile (finis "
 do while queued() > 0
    pull line
    /*(process records as required) */
 end
 "free dd(infile)"
```

Allocate the data set to the program as a ddname of INFILE. Fully qualified data set names must be enclosed in single quotes. As the file is only being read, the SHR disposition is being used. The REUSE parameter is coded so if the program fails before freeing it, the allocate command will not fail when the program is rerun. As with all REXX instructions, uppercase is optional.

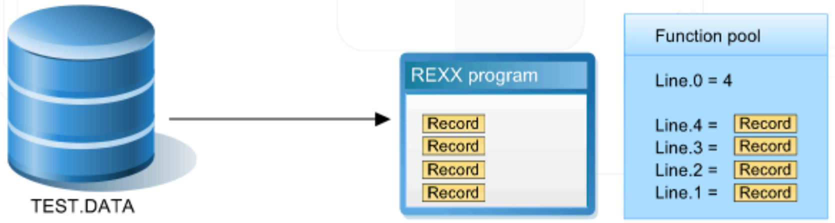
This section of code shows how the ALLOC and EXECIO commands can be used to process all the records from a data set and place them in the stack.

Click **Play** to see the effect that each section of code has on the data set.



```
EDIT       USER1.REXX.EXEC(READ2STM)                        Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Dat
/* REXX - This program read records into
Address TSO
"alloc dd(infile) da('test.data') shr re

"execio * diskr infile (STEM LINE.finis
 do ll = 1 to line.Ø
    /*(process records as required) */
 end
 "free dd(infile)"
```

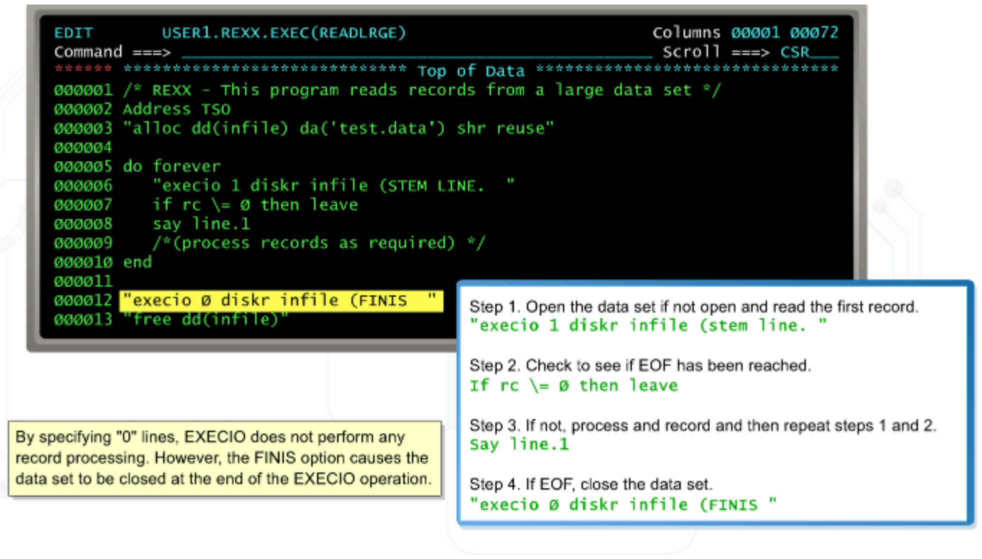
The records can now be processed by using standard iterative DO/END group logic. Variables are still set to their respective records and can be reprocessed as often as necessary, unlike using the stack.



More commonly, records from data sets are saved into compound variables. This is achieved by specifying the STEM varname option; varname defines the prefix of a simple or compound variable name to be used, which is followed by the number of the record extracted by this operation. The variable varname0 is set to the total number of records extracted in this operation. The variables are treated as compound variables if varname ends in a ".".

Technically, the return code from the EXECIO command should always be checked to ensure the operation worked successfully. Valid return codes are: 0 - Successful completion; 1 - Data was truncated; 2 - EOF reached; 4 - Empty data set; 20 - Severe error.

Click **Play** to see an example of EXECIO reading into stem variables.



```
EDIT       USER1.REXX.EXEC(READLRGE)                    Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
/* REXX - This program reads records from a large data set */
Address TSO
"alloc dd(infile) da('test.data') shr reuse"

do forever
   "execio 1 diskr infile (STEM LINE.  "
   if rc \= 0 then leave
   say line.1
   /*(process records as required) */
end

"execio 0 diskr infile (FINIS  "
"free dd(infile)"
```

Step 1. Open the data set if not open and read the first record.
`"execio 1 diskr infile (stem line. "`

Step 2. Check to see if EOF has been reached.
`If rc \= 0 then leave`

Step 3. If not, process and record and then repeat steps 1 and 2.
`Say line.1`

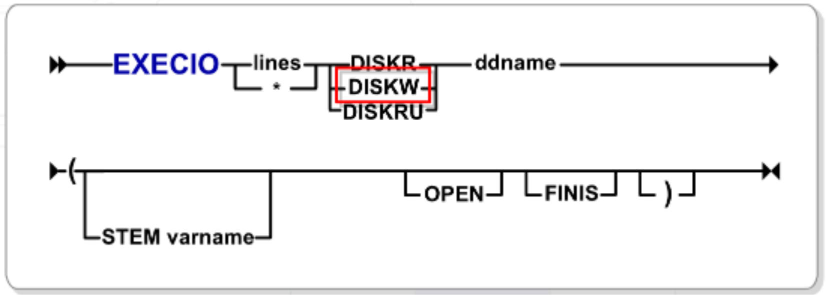
Step 4. If EOF, close the data set.
`"execio 0 diskr infile (FINIS "`

By specifying "0" lines, EXECIO does not perform any record processing. However, the FINIS option causes the data set to be closed at the end of the EXECIO operation.

Sometimes the data set to be read contains more records than can be stored in the address space being used by the REXX program. This typically exceeds 100,000 records, depending on the data center's standards, so records can only be read in smaller numbers and, for logical purposes, one at a time. The stack or stem variables can be used. The steps in this process are:

1. Open the data set if not open and read the first record.
2. Check to see if EOF has been reached.
3. If not, process record and then repeat steps 1 and 2.
4. If EOF, close the data set.

Click Play to see how to perform these actions.



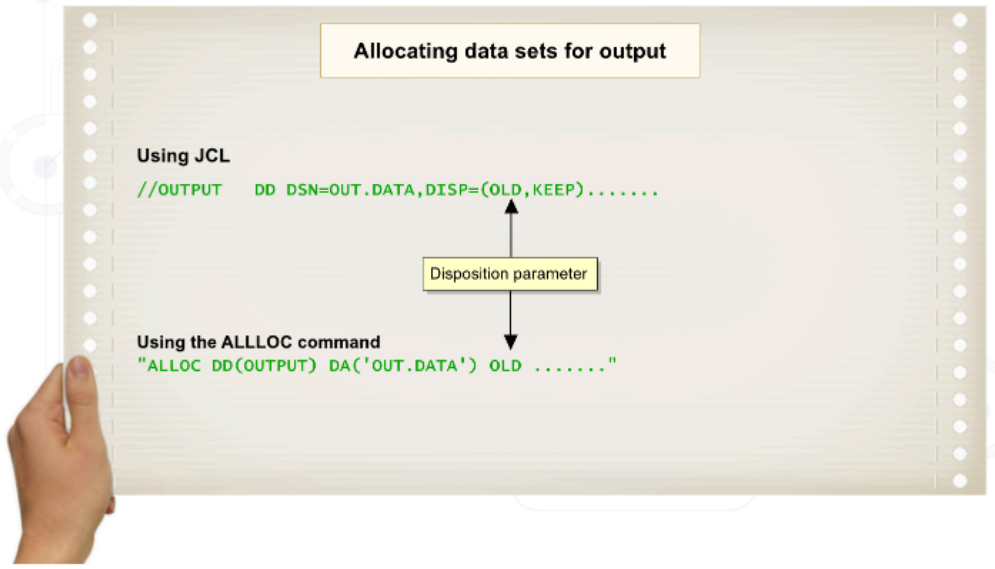
The DISKW parameter indicates that a "write" or "output" operation is to be performed in the data set.

The linenum parameter and FIFO, LIFO, and SKIP options have no meaning or effect when performing DISKW operations.

The DISKW parameter is used to write to a data set in much the same way as DISKR is used to read one. Records can be written from the stack or compound variables.

However, some EXECIO options have no effect when using DISKW and the "*" (arbitrary lines) parameter can cause problems if used.

Click **Play** to see the parameters that have no effect when using the DISKW parameter.



Allocating data sets for output

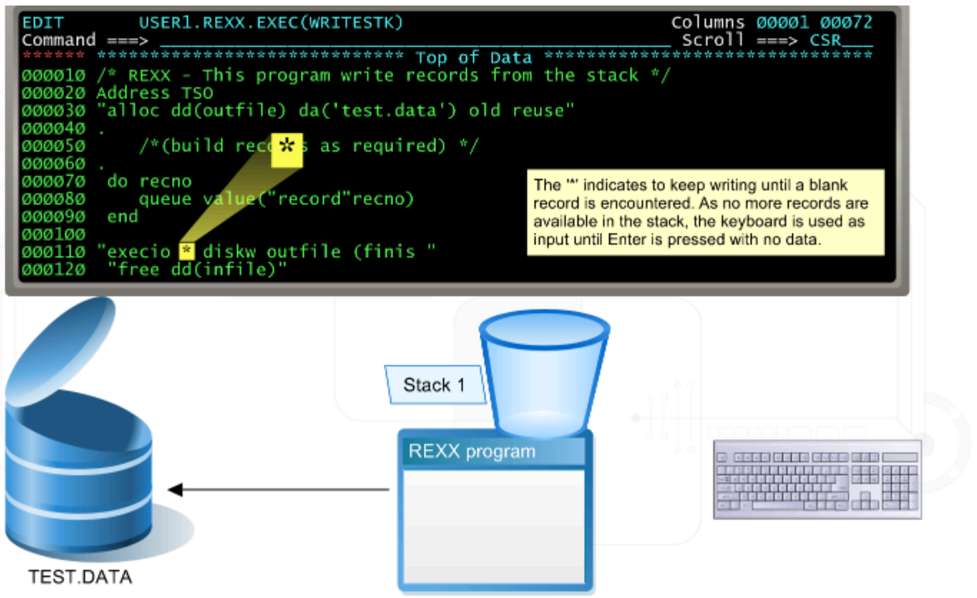
Using JCL

```
//OUTPUT   DD DSN=OUT.DATA,DISP=(OLD,KEEP).......
```

Using the ALLLOC command

```
"ALLOC DD(OUTPUT) DA('OUT.DATA') OLD ......."
```

When allocating a data set for output, take care when defining the data set disposition regardless of whether the data set is allocated by using the ALLOC command or JCL. The implications of each disposition status should be clearly understood:
NEW - Indicates the data set is being created and used for the first time.
SHR - Indicates the data can be used by other users at the same time. This is not recommended when writing to a data set as data corruption may occur; recommended for read operations only.
OLD - Indicates the data set already exists and should be dedicated to this operation. If the data set is opened as "output" for writing, any existing data will be overwritten.
MOD - Indicates that any data written to this data set will be added to the end of the data set. When opened, the CRP will be placed at



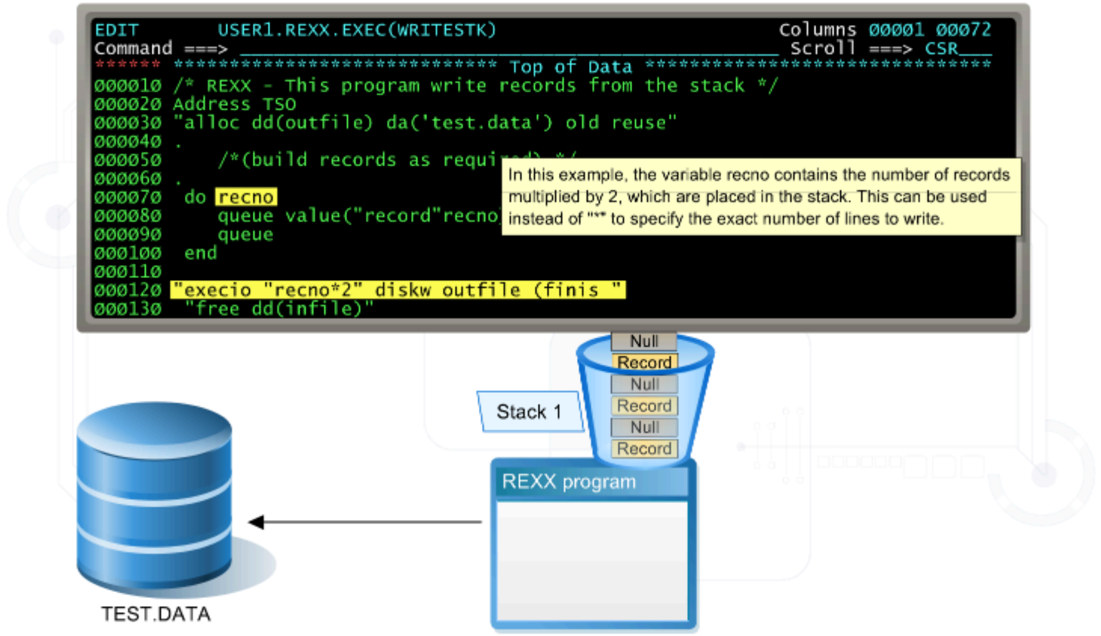
```
EDIT       USER1.REXX.EXEC(WRITESTK)                        Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
/* REXX - This program write records from the stack */
Address TSO
"alloc dd(outfile) da('test.data') old reuse"
.
    /*(build records as required) */
.
 do recno
    queue value("record"recno)
 end

"execio * diskw outfile (finis "
 "free dd(infile)"
```

The "*" indicates to keep writing until a blank record is encountered. As no more records are available in the stack, the keyboard is used as input until Enter is pressed with no data.

If the STEM option on the EXECIO command is not specified when writing records to a data set, it will get the records from the stack.

When the "*" parameter is used to specify the number of lines to be written, EXECIO keeps writing until it encounters either a null line or an uninitialized compound variable. If all records have been written and no null lines or uninitialized variables were encountered, EXECIO waits for input from the keyboard and continues to write any data typed when Enter is pressed. It does this until Enter is pressed without any data.

Click Play to see an example of records being written from the stack.



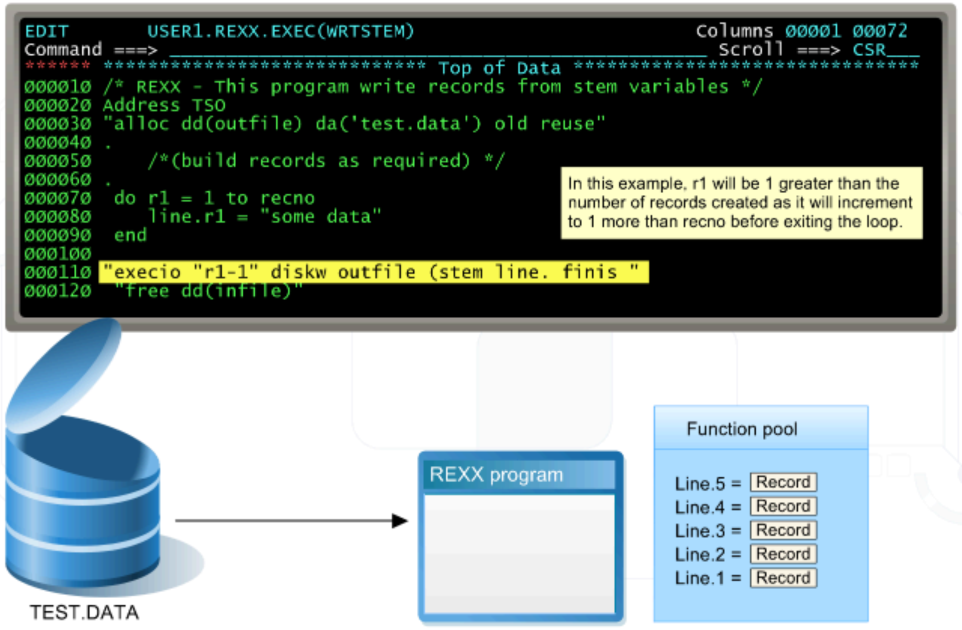
```
EDIT       USER1.REXX.EXEC(WRITESTK)                     Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** **************************** Top of Data ******************************
000010 /* REXX - This program write records from the stack */
000020 Address TSO
000030 "alloc dd(outfile) da('test.data') old reuse"
000040 .
000050    /*(build records as requi
000060 .
000070  do recno
000080    queue value("record"recno
000090    queue
000100  end
000110
000120 "execio "recno*2" diskw outfile (finis "
000130  "free dd(infile)"
```

In this example, the variable recno contains the number of records multiplied by 2, which are placed in the stack. This can be used instead of "*" to specify the exact number of lines to write.

Problems can also occur if blank records are required to be written before the end of all the data.

It is recommended that a counter is kept within the REXX code for each time a record is created for writing, and the counter is used in the EXECIO command instead of "*".

Click Play to see an example of records being written from the stack.



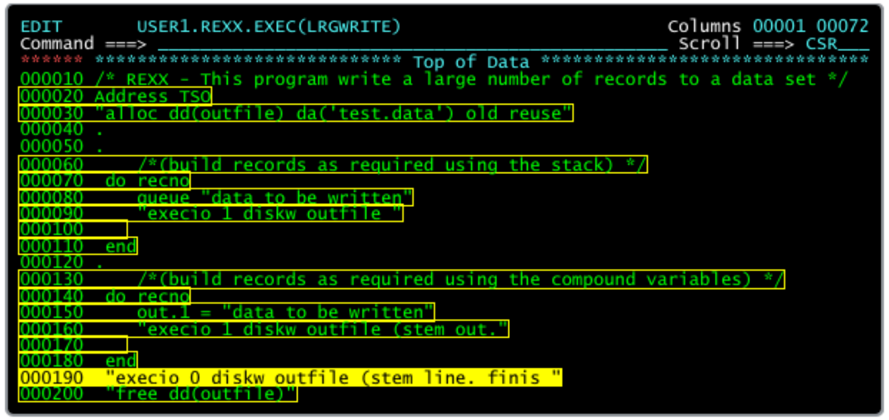
```
EDIT       USER1.REXX.EXEC(WRTSTEM)                         Columns 00001 00072
Command ===> ______________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
000010 /* REXX - This program write records from stem variables */
000020 Address TSO
000030 "alloc dd(outfile) da('test.data') old reuse"
000040 .
000050    /*(build records as required) */
000060 .
000070  do r1 = 1 to recno
000080     line.r1 = "some data"
000090  end
000100
000110 "execio "r1-1" diskw outfile (stem line. finis "
000120  "free dd(infile)"
```

In this example, r1 will be 1 greater than the number of records created as it will increment to 1 more than recno before exiting the loop.

As an alternative to the stack, compound variables can be used to store records and write them to a data set. Again, a counter should be used when writing records to a data set.

To use compound variables, the STEM varname option should be used on the EXECIO command where varname is the compound variable stem.

Click **Play** to see how compound variables are used for writing data to data sets.



```
EDIT       USER1.REXX.EXEC(LRGWRITE)                           Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
/* REXX - This program write a large number of records to a data set */
Address TSO
"alloc dd(outfile) da('test.data') old reuse"
.
.
    /*(build records as required using the stack) */
 do recno
    queue "data to be written"
    "execio 1 diskw outfile "

 end
.
    /*(build records as required using the compound variables) */
 do recno
    out.1 = "data to be written"
    "execio 1 diskw outfile (stem out."

 end
 "execio 0 diskw outfile (stem line. finis "
 "free dd(outfile)"
```

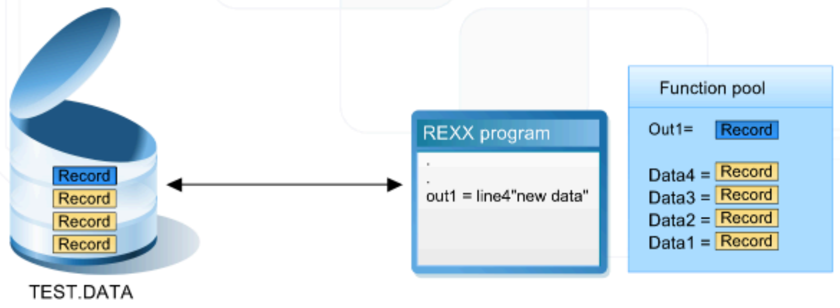
After writing all required records, the data set must be closed. As no more records are required to be written, the lines specified is 0 with the FINIS option included.

When large amounts of data are required to be written to a data set, and the number of records greatly exceeds 100,000, storage restraints may prevent them from being stored either in the stack or in compound variables.

In this case, it is usually necessary to write a single record at a time. The above example shows some code that could be used to write single records to a data set.

Mouse-over the code for descriptions of each section.

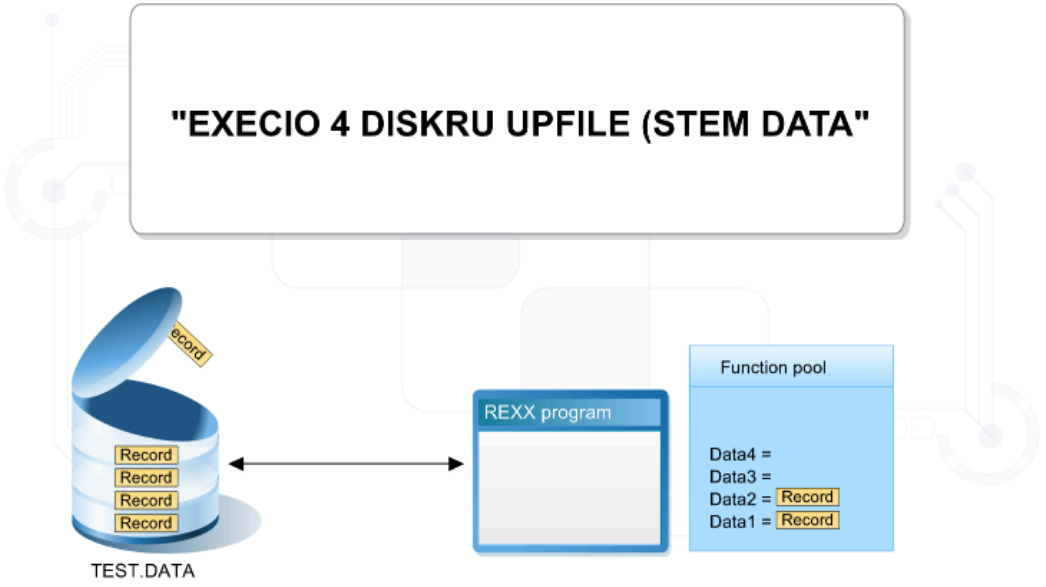
"EXECIO 1 DISKW UPFILE (STEM OUT "



The DISKRU parameter enables the data set to be both read and written to. When the data set is read, the last record to be retrieved can be changed by executing a write action using DISKW. Any write action will replace the last record read.

This is the only situation where different DISKx parameters can be used on the same data set without closing it first. Because only one record can be updated at a time, DISKRU operations are usually performed singularly, that is, record by record. The stack or the compound variable can be used to store the records.

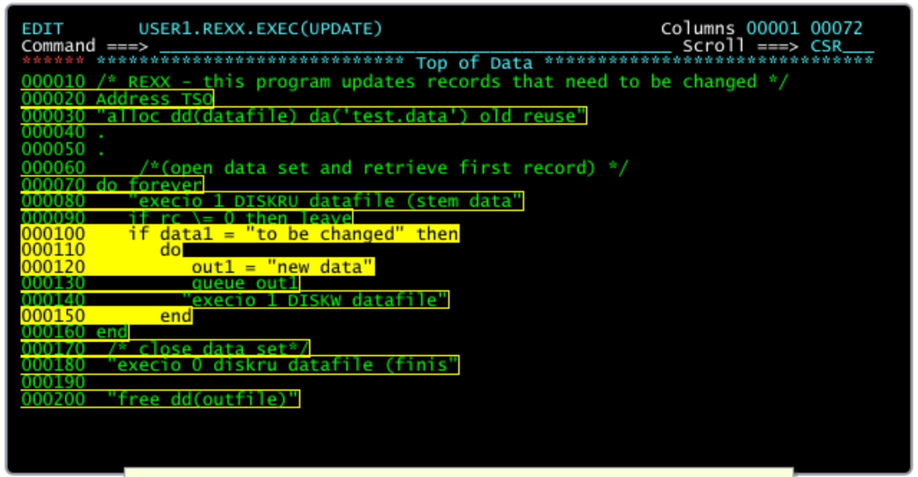
Click Play to see an example of the DISKRU operation.



The DISKRU parameter enables the data set to be both read and written to. When the data set is read, the last record to be retrieved can be changed by executing a write action using DISKW. Any write action will replace the last record read.

This is the only situation where different DISKx parameters can be used on the same data set without closing it first. Because only one record can be updated at a time, DISKRU operations are usually performed singularly, that is, record by record. The stack or the compound variable can be used to store the records.

Click **Play** to see an example of the DISKRU operation.



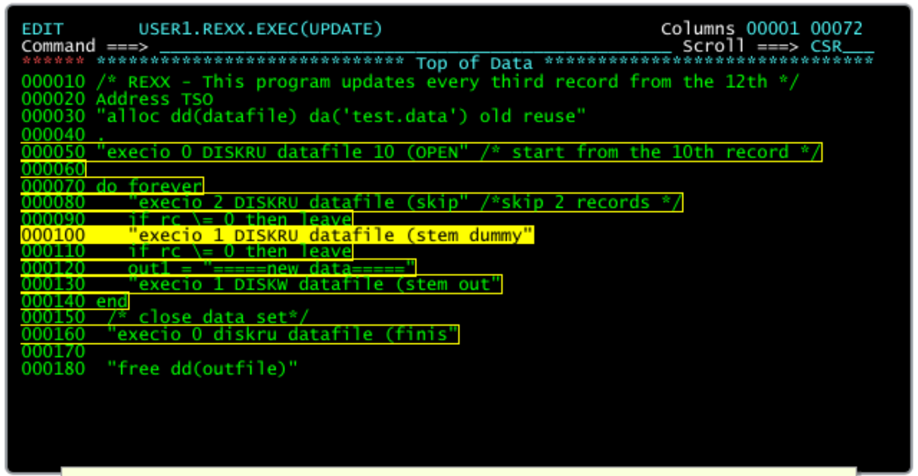
```
EDIT       USER1.REXX.EXEC(UPDATE)                     Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
/* REXX - this program updates records that need to be changed */
Address TSO
"alloc dd(datafile) da('test.data') old reuse"
.
.
    /*(open data set and retrieve first record) */
do forever
   "execio 1 DISKRU datafile (stem data"
   if rc \= 0 then leave
   if data1 = "to be changed" then
      do
         out1 = "new data"
         queue out1
        "execio 1 DISKW datafile"
      end
end
 /* close data set*/
 "execio 0 diskru datafile (finis"

 "free dd(outfile)"
```

This conditional statement determines if the record is to be changed. If so, it sets the new value. As the variable name and the actual record are not physically linked, the new value could be in the same variable or a new one.

This example shows how records can be read one at a time and changed as required. A data set that is to be updated should be allocated a disposition of OLD because a NEW data set has no records to be updated. MOD will not work as the CRP is placed at the end of the data set, and a read operation will only receive an end of file (EOF). Records cannot be added to a data set by using DISKRU.

Notice that the data set is read in the same way that a large data set would be read. The final EXECIO operation to close the data set can be a DISKRU or a DISKW operation.

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



```
EDIT       USER1.REXX.EXEC(UPDATE)                      Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
/* REXX - This program updates every third record from the 12th */
Address TSO
"alloc dd(datafile) da('test.data') old reuse"
.
"execio 0 DISKRU datafile 10 (OPEN" /* start from the 10th record */

do forever
   "execio 2 DISKRU datafile (skip" /*skip 2 records */
   if rc \= 0 then leave
   "execio 1 DISKRU datafile (stem dummy"
   if rc \= 0 then leave
   out1 = "=====new data====="
   "execio 1 DISKW datafile (stem out"
end
 /* close data set*/
 "execio 0 diskru datafile (finis"

 "free dd(outfile)"
```

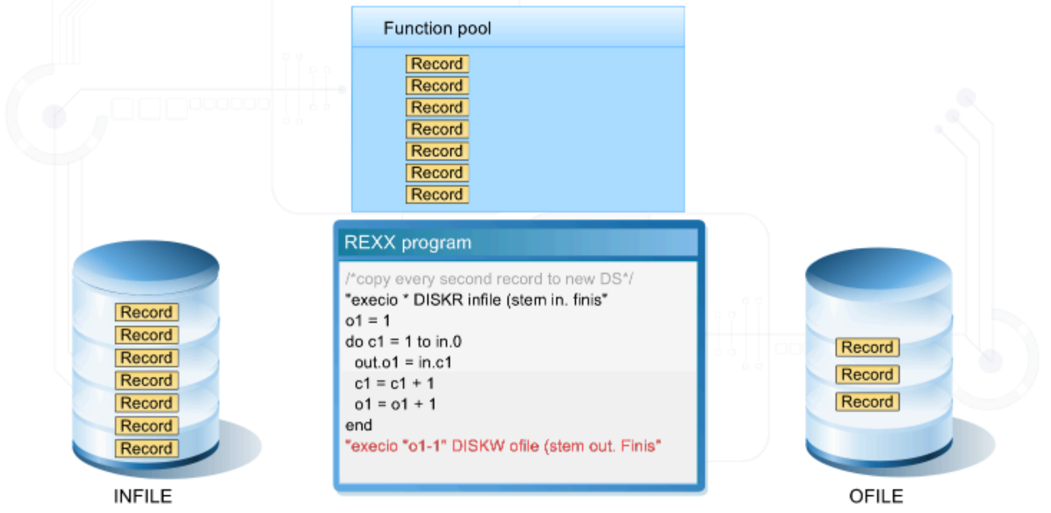
Read the next record. For the first iteration of this loop, this record would be record 12. The record will be read into a variable called DUMMY1. This variable will not be used again.

In some instances, the specific record number that must be updated in the data set is known. Using the previous method, however, all earlier records would have to be read first. The LINENUM parameter enables a DISKRU or DISKR operation to start at a specific record.

Alternatively, the SKIP option can be used to skip a number of lines without placing them in the stack or variables. The next DISKR or DISKRU operation reads the following line in the data set. For example, if 23 lines are skipped when the data set is first opened, the next read operation extracts the 24th record.

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

Copying selected records

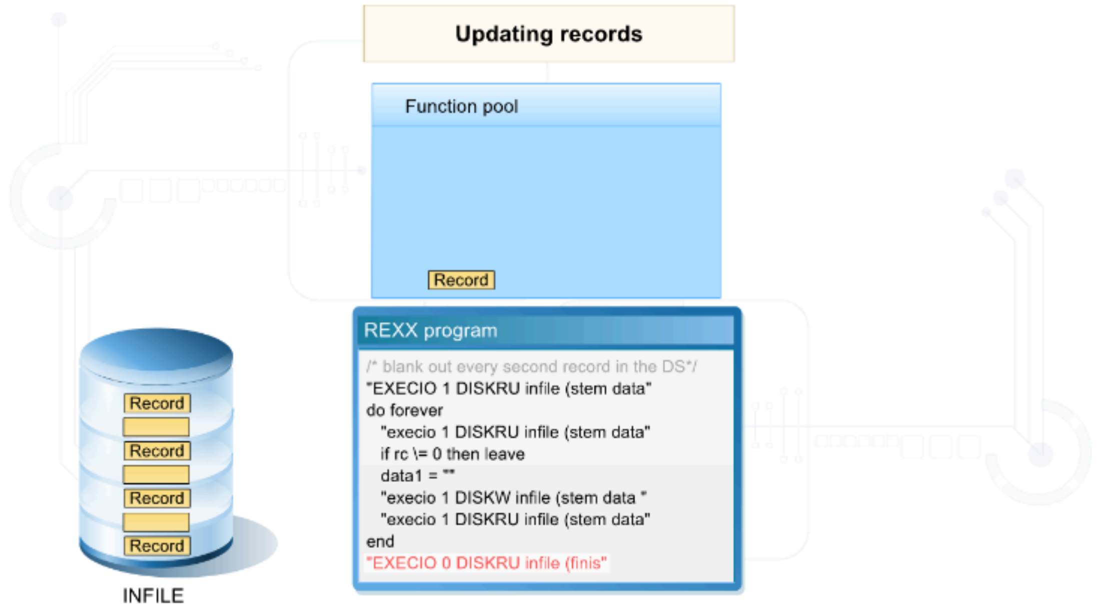


```
/*copy every second record to new DS*/
"execio * DISKR infile (stem in. finis"
o1 = 1
do c1 = 1 to in.0
  out.o1 = in.c1
  c1 = c1 + 1
  o1 = o1 + 1
end
"execio "o1-1" DISKW ofile (stem out. Finis"
```

The EXECIO command cannot delete individual records by itself. It can only be used to read all the records and copy back the required records. This example shows how one file can be read and selected records written out to another. The original file can then be deleted and the new one renamed.

Alternatively, all the records can be read first by a DISKR operation and, after closing the data set, a DISKW operation can be used to rewrite the data in the original file.

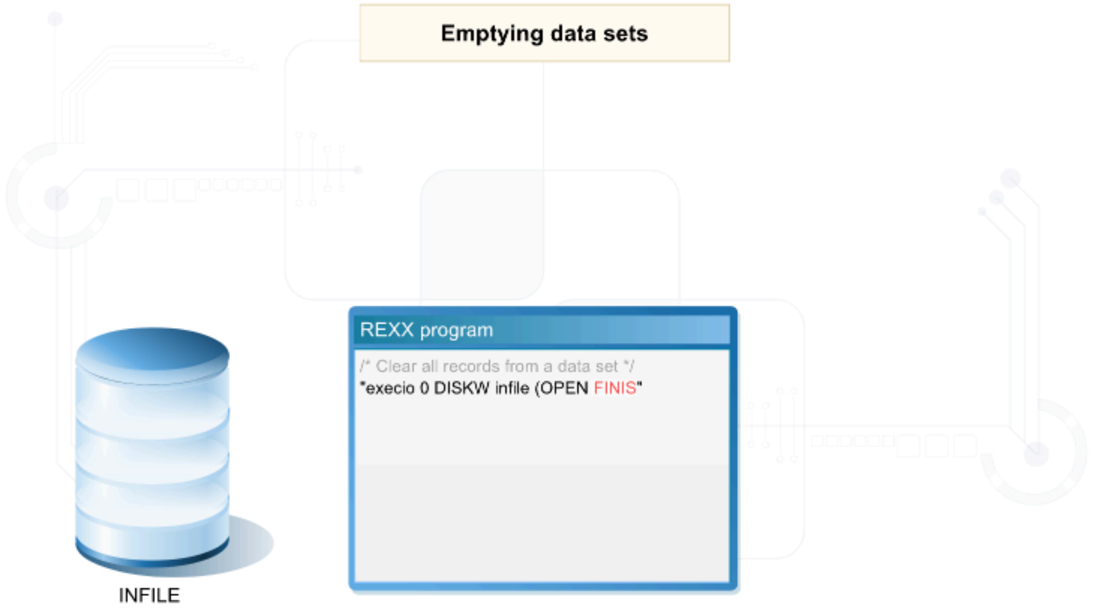
Click Play to see how selected records can be written to a file.



```
/* blank out every second record in the DS*/
"EXECIO 1 DISKRU infile (stem data"
do forever
  "execio 1 DISKRU infile (stem data"
  if rc \= 0 then leave
  data1 = ""
  "execio 1 DISKW infile (stem data "
  "execio 1 DISKRU infile (stem data"
end
"EXECIO 0 DISKRU infile (finis"
```

The EXECIO command can be used to blank out records by using DISKRU and DISKW operations to set the updated records to blank; however, no records will actually be deleted.

Click Play to see how selected records can be set to blanks.



```
/* Clear all records from a data set */
"execio 0 DISKW infile (OPEN FINIS"
```

The DISKW operation can also remove all the records from a data set by opening it in write mode and closing it without writing any records.

Click **Play** to see how the DISKW operations, in conjunction with the OPEN and FINIS options, will clear a data set.