Executing REXX Programs in TSO/E

Objectives

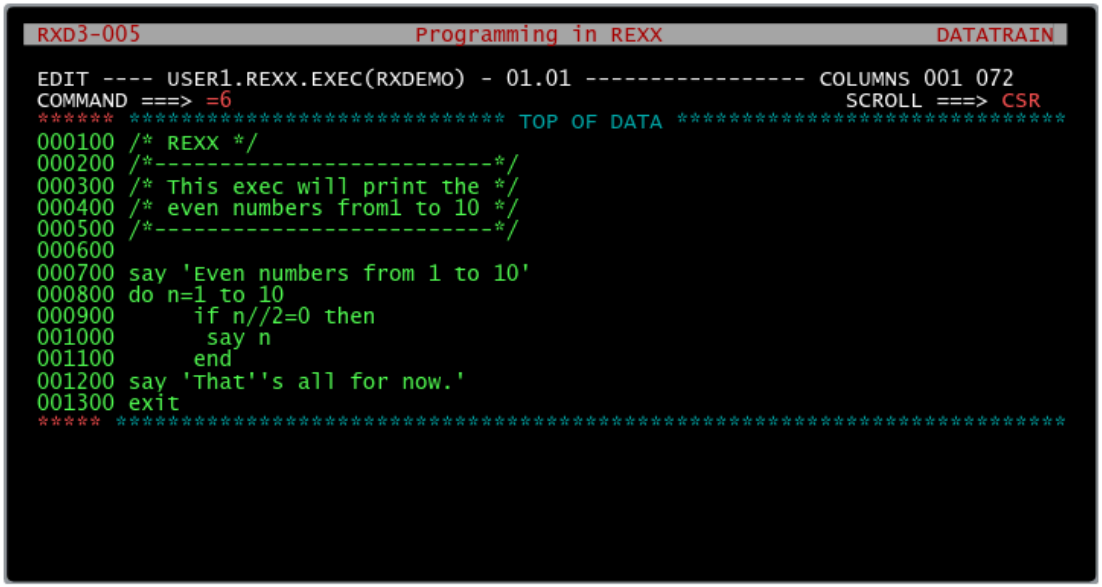
Executing REXX Programs in TSO/E

In this module, you will be introduced to some basic REXX code and see how it is executed in a mainframe system environment.

You will examine the processes that the code performs and review some common REXX functions and keywords.

After completing this module, you will be able to:

- Execute REXX Programs in an IBM Mainframe Environment
- Identify Some Common REXX Functions and Keywords

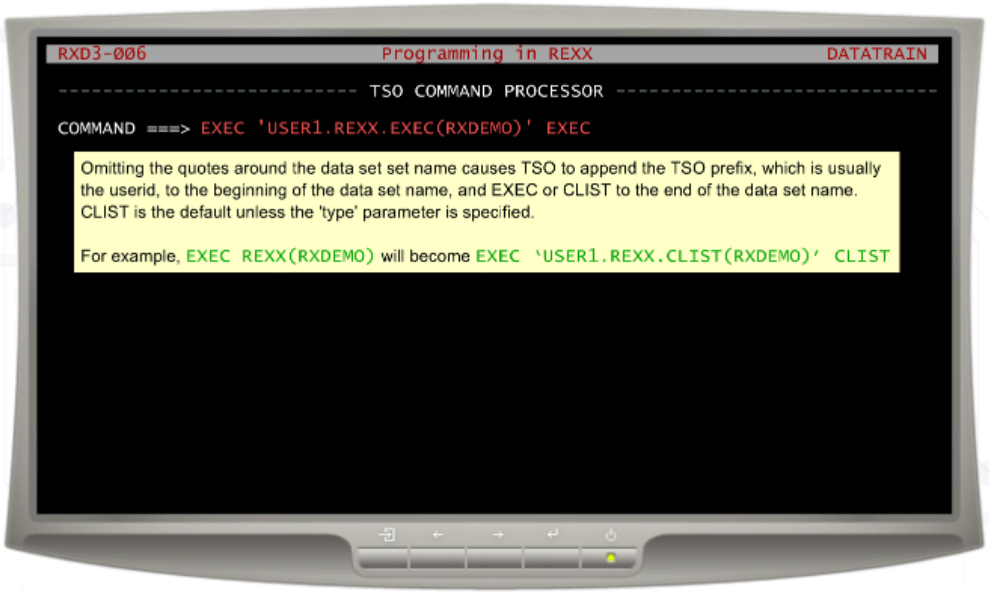


This is a simple REXX program, which is also called an EXEC. It is defined in a z/OS environment under TSO/E and the ISPF editor.

EXECs are usually executed by a TSO command from the ISPF command line or from the TSO command processor.

Type =6 on the command line to access the TSO command processor.

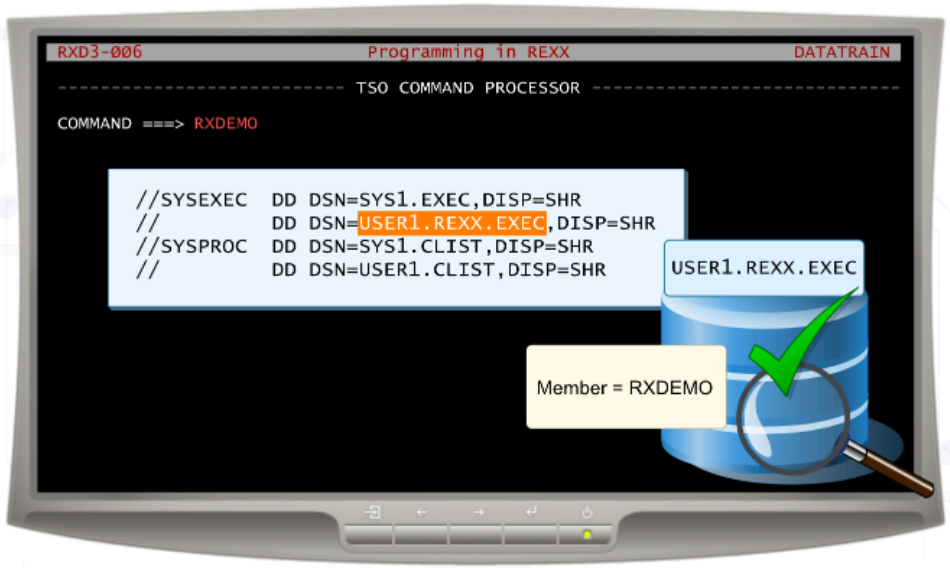
Press **Enter** when you have finished.



REXX execs can be run explicitly by using the EXEC TSO command with a fully qualified data set name and member.

Use the EXEC parameter after the data set name to distinguish the REXX exec from a CLIST.

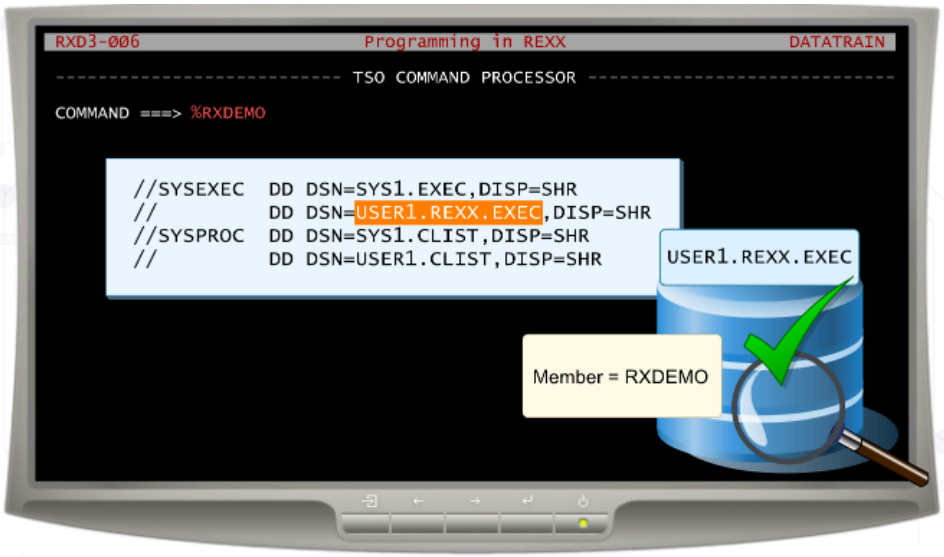
Click Play to see the syntax of the EXEC TSO command and how it is used.



Alternatively, a REXX exec can be executed implicitly by using the member name of the REXX exec, but the data set containing the REXX program in that member must be allocated to the DD name concatenation SYSEXEC, which is only used for EXECS, or SYSPROC, which is used for both CLISTS and EXECS. Note that REXX programs in data sets defined to SYSPROC must start with a comment containing the word "REXX".

When the REXX member name is executed, TSO searches for a program or command of that name, and then for the SYSEXEC and SYSPROC DD name concatenation libraries.

Click **Play** to see the standard search order used by TSO for implicit execution.



Occasionally, the name of the member in which the REXX program resides is the same as a TSO command or program, which causes confusion when you attempt to run it.

Preceding the member name with a percent sign (%) causes TSO to look only in the SYSEXEC and SYSPROC DD definition concatenations for the member containing the REXX program. This is called extended implicit execution.

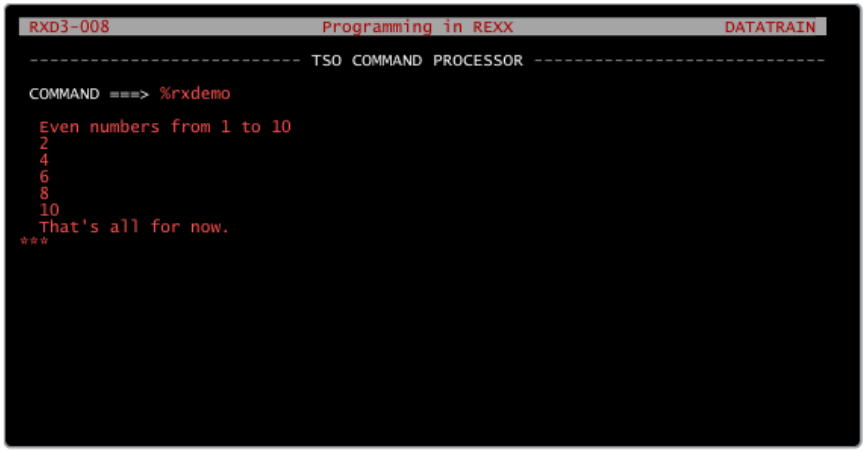
Click **Play** to see the search order used by TSO when the implicit execution command is preceded by %.

An alternate library can be added to the search order prior to the SYSEXEC and SYSPROC definitions by using the ALTLIB command.

Defining an ALTLIB enables REXX program libraries to be easily added to the search order for specific applications and then removed after the application has completed.

Click **Play** to see how the ALTLIB command is used to alter the search order for implicitly called REXX programs.

Click **Play** to see the RXDEMO REXX executed and its results.



You will now take a step-by-step look at the REXX code that causes each line of the display to occur.

When the command %RXDEMO command is entered, TSO searches the REXX and CLIST libraries for the first occurrence of a member named RXDEMO.

When this is found, TSO opens the member and begins to interpret the program contained in the member.

Type %RXDEMO on the command line to start the REXX program.

Press **Enter** when you have finished.

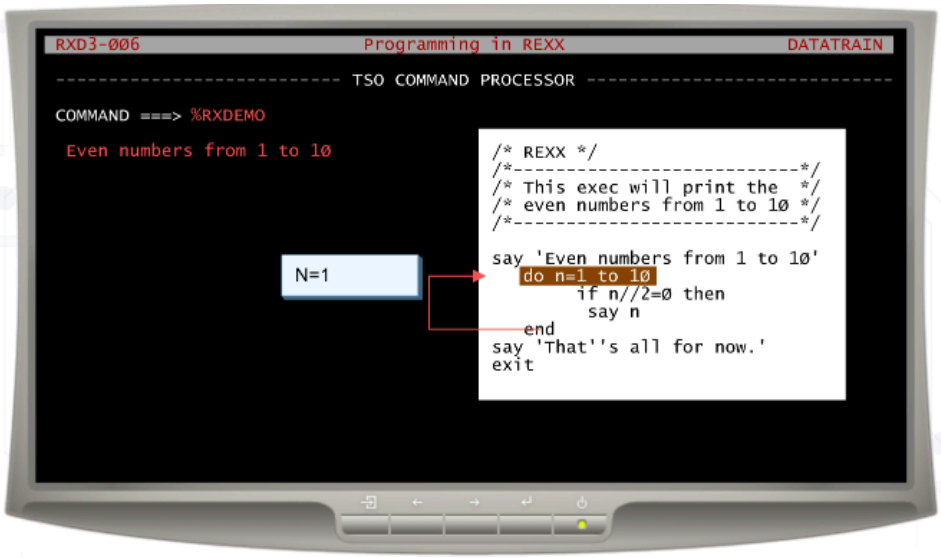
The first line identifies the EXEC to the systems interpreter. Because this line has a comment containing the word "REXX", the system invokes the REXX interpreter rather than the CLIST interpreter.

Click Play to see the code and the REXX comment.

The comment lines enclosed by /* and */ are ignored and the interpreter goes to the first executable statement.

The SAY keyword instruction causes the text following it to be displayed on the screen.

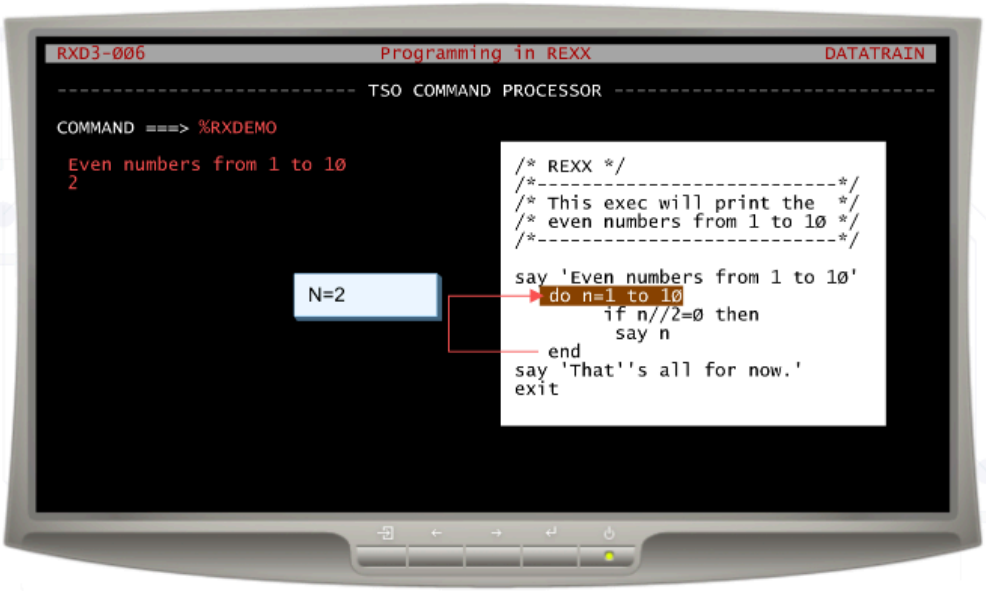
Click **Play** to see the REXX program continuing.



The next statement to be executed is the entry to a DO loop. The `do` keyword instruction defines the beginning of a set of statements through to the `end` keyword instruction, which will loop under defined conditions.

In this instance, the `do` keyword sets the value to 1 to be stored in the `n` variable. Each time the loop executes, this `n` variable will be incremented by 1 until it reaches the value of 11. In the `if` keyword, the remainder (the `//` operator - two slashes together) of `n` divided by 2 is checked to see if it is 0. Since this remainder is 1, nothing happens. Processing continues until the `end` keyword instruction then loops back to the `do` statement.

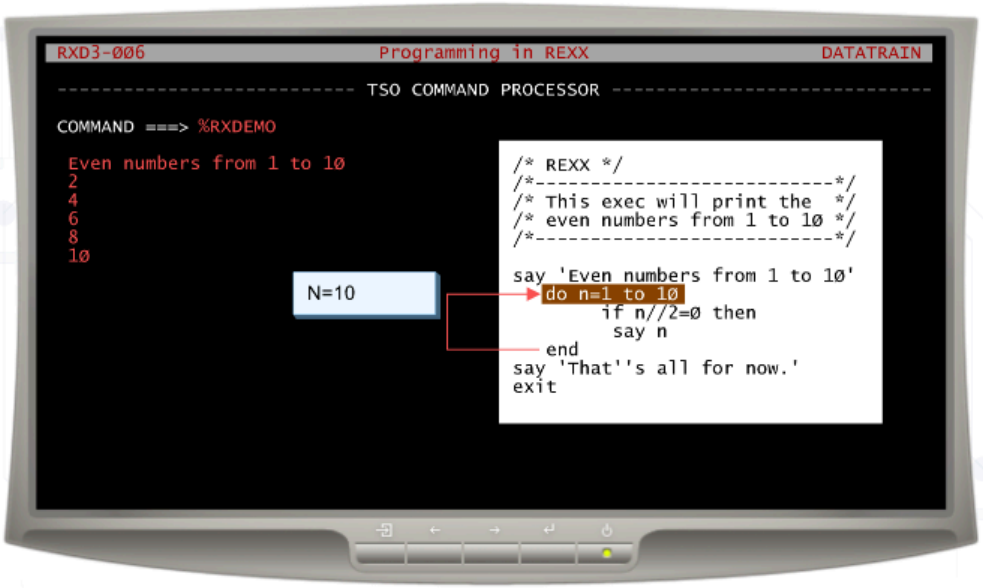
Click Play to see how the `do` loop code processes.



The DO loop increments the value of `n` by 1 and tests to see if it is greater than 10. Since `n` is 2, the loop continues.

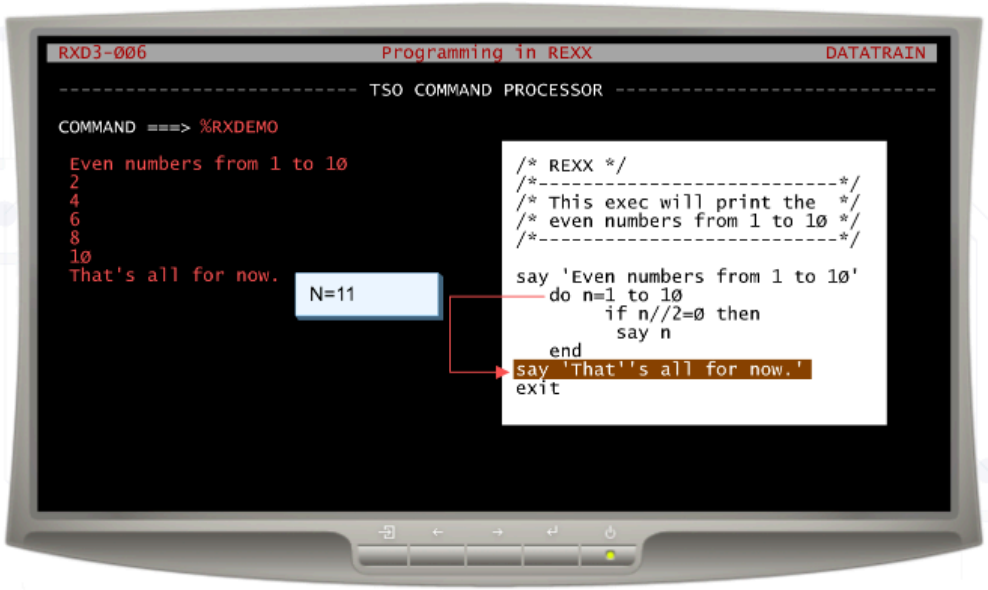
This time, n//2 returns 0 so the `if` expression evaluates to true and the `then` statement executes. The `say` statement tells REXX to write the value of `n` onto the terminal display.

Click Play to watch the DO loop code continue processing.



Processing continues until n contains the value of 10. The program will now loop back once more to the do statement.

Click Play to continue the REXX processing.



When n is incremented this time, it has the value 11. Because this is greater than 10, the loop is terminated and processing continues from the next statement after the loop.

The say keyword instruction causes the text in quotes to be written to the terminal. The two single quotes coded together ('') are translated to a single quote (').

Click Play to see REXX processing the end of the loop.

Finally, the `exit` statement tells REXX to terminate the exec.

Control is then returned to TSO.

Click Play to see the REXX program terminating.