UP NEXT: Rexx

Enterprise IT's Duct Tape

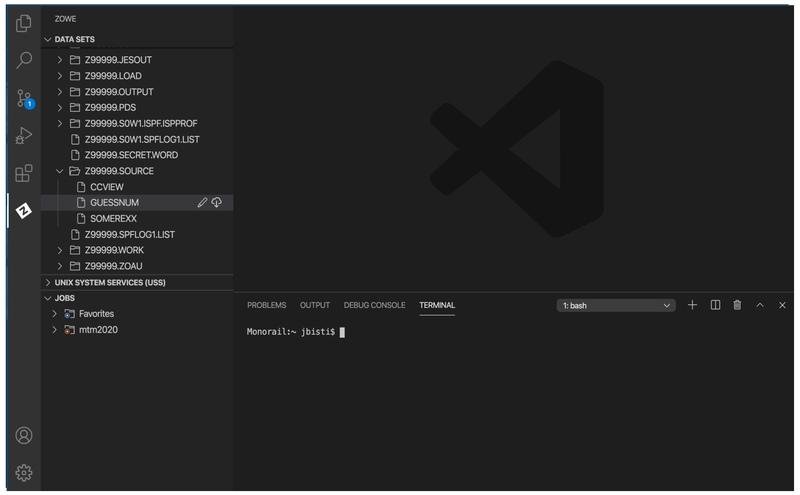
12 steps 3 hours

THE CHALLENGE

In this challenge, you'll meet Rexx, a programming language known for its simplicity, power, and relative ease of use. We'll dive into how you run a Rexx program from a command line, as well as how to start up a TSO Address Space to run an interactive Rexx program.

BEFORE YOU BEGIN

This challenge requires some of the configuration done in the ZCLI1 challenge. If you haven't done that, complete ZCLI1 (or at least the first 3 steps) and then re-try this.

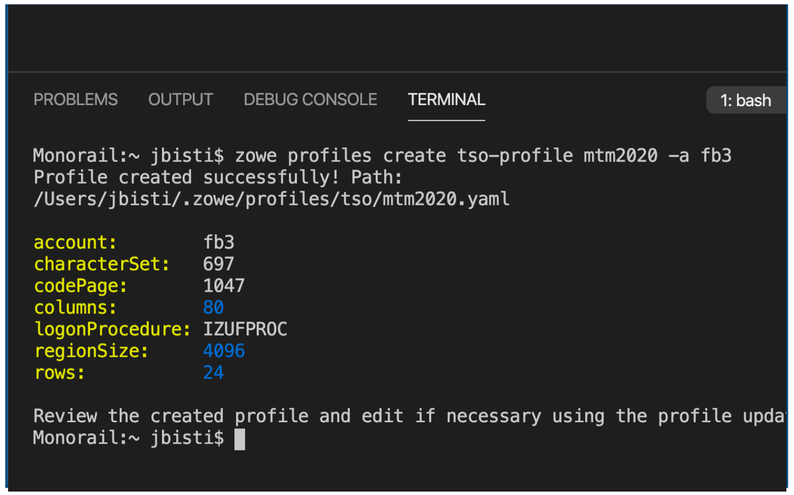


1. OPEN A TERMINAL

Start by copying the two files from MTM2020.PUBLIC.SOURCE into your own SOURCE data set. Specifically, we're looking for SOMEREXX and GUESSNUM.

Next, open up a Terminal, just like you did for the USS commands, but don't SSH into anything. You should be able to type the command zowe from here and get some output from the program

Important Note: If the 'zowe' command does not work, go back to the ZCLI1 challenge and double-check those required first 3 installation steps for the Zowe CLI challenge.

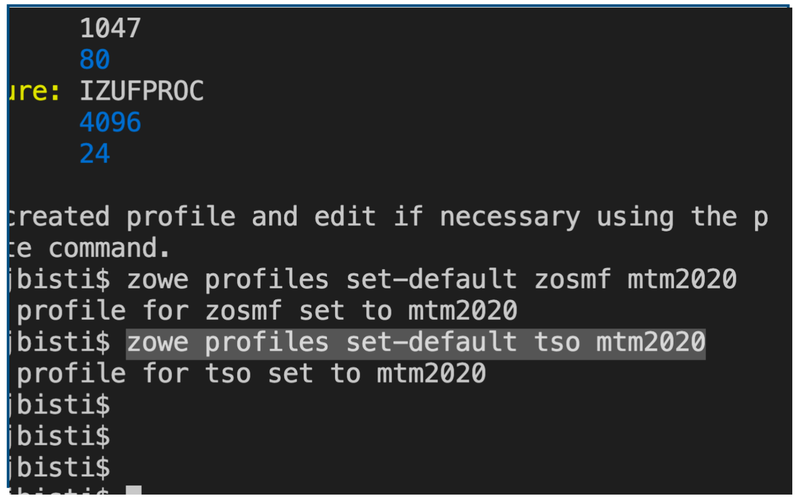


2. CREATE A TSO PROFILE

So far, we've been using a z/OSMF profile to connect. We need to create a TSO profile to issue TSO commands (more info on next page). Type this command:

zowe profiles create tso-profile mtm2020 -a fb3

Use whatever profile name you want, mtm2020 is just an example, and if you've been following our examples, it's nice to stay consistent.

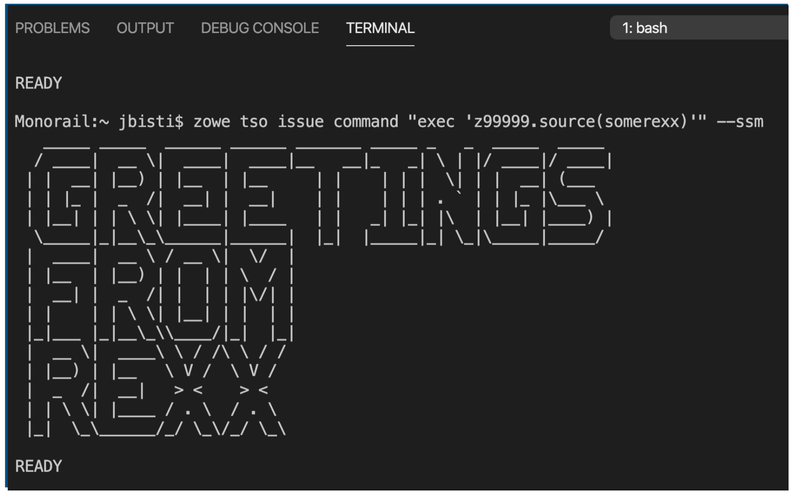


3. SET DEFAULT PROFILES

At this point, it's also a good idea to make sure the right profiles are set as the defaults. They probably are, but just in case:
zowe profiles set-default zosmf mtm2020
zowe profiles set-default tso mtm2020

(Of course, use the profile names you chose)

If you want to start over, use the **zowe profiles delete** command and follow the prompts.

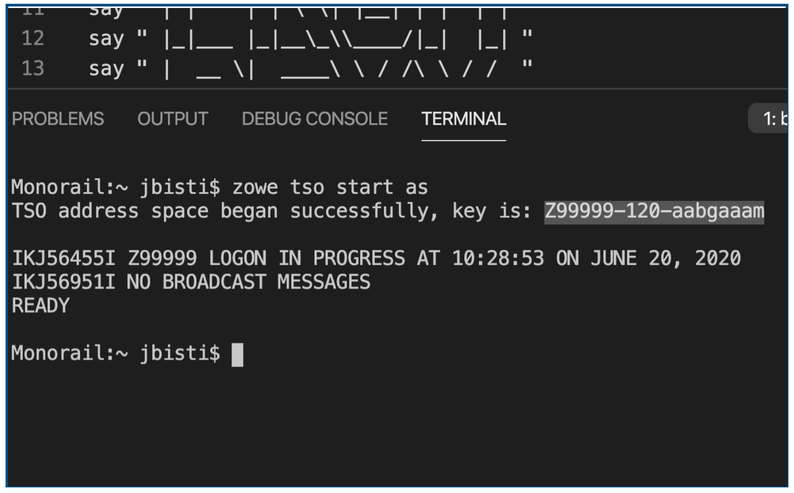


4. RUN SOME Rexx

Type the following command:

zowe tso issue command "exec 'zXXXXX.source(somerexx)'" --ssm

Make sure to include all of the double and single quotes. Also, this will be the last time we point out that whenever you see Zxxxxx or Z99999, you need to input your own userid. We know most of you are tired of seeing that spelled out every time, but you wouldn't believe how many people get confused by not replacing sample text.

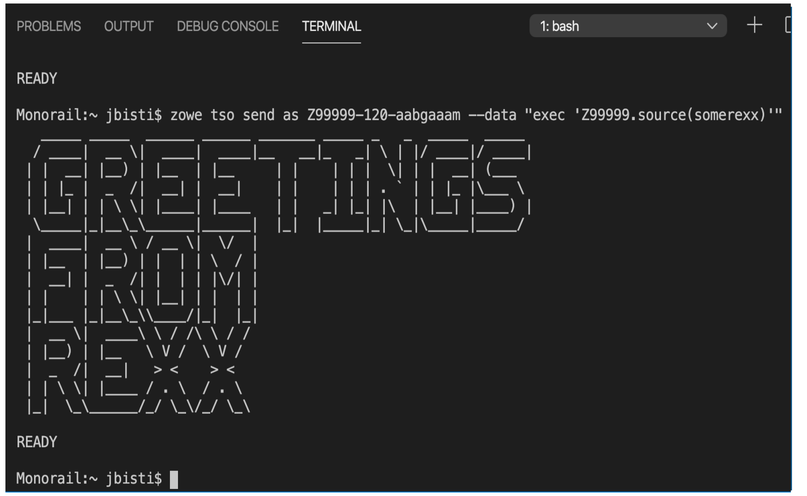


5. START AN ADDRESS SPACE

Start an address space with the following command:
zowe tso start as

This will create an address space for you, and tell you its key, which will start with your userid (as you can see above). You will use this key for the next few steps. Sometimes, after not being used for a while, a TSO session goes away. Just make another one using the same command.

You can stop an address space with the **stop** option



6. RUN THAT SAME REXX

Run the same Rexx program we did in step 4, but this time, direct the input towards the address space (and remember, you can probably press the Up arrow to recall previous commands)

zowe tso send as your-as-key --data "exec 'Zxxxxx.source(somerexx)'"

Note: (1) That's all on one line
(2) your key will be different
(3) this time we leave off the --ssm

You should get back the exact same response as before. The difference is that you're now issuing these commands through a semi-persistent TSO Address Space, which will make more sense in the next step.

"TSO? ADDRESS SPACE?"

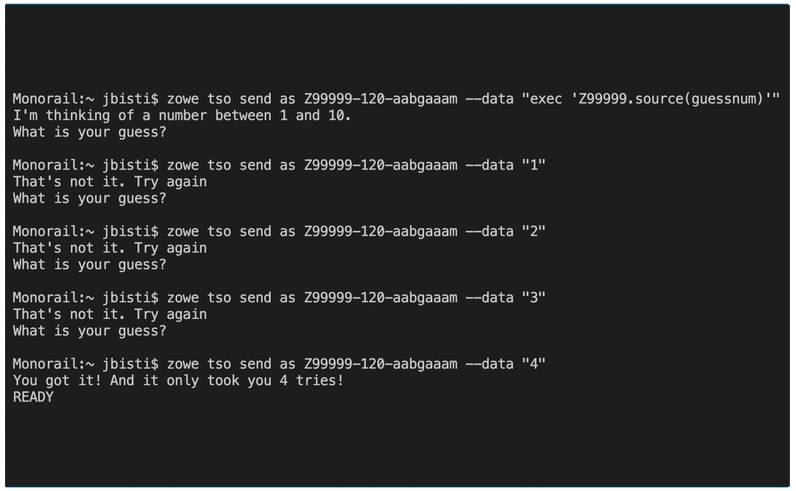
TSO (Time Sharing Option) is another way that z/OS allows many many users to get access to data sets, run programs, and look at output. It is essentially the command-line interface for z/OS (when you're not using USS to access the UNIX side of things)

Think of an Address Space as a ticket that lets you start using system memory. An address space represents an enormous amount of memory, though the system will actually still control what lives in real on-chip memory, vs what gets moved (or *paged*) out to disk. Where your program goes its memory depends on how important it is, how it should be used, and if it will be shared with other programs. Address Spaces are a core part of z/OS, and you should read more about them when you get a minute: https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zconcepts/zconcepts_82.htm

7. RUN INTERACTIVE REXX

Do the same, but with the program ‘GUESSNUM’

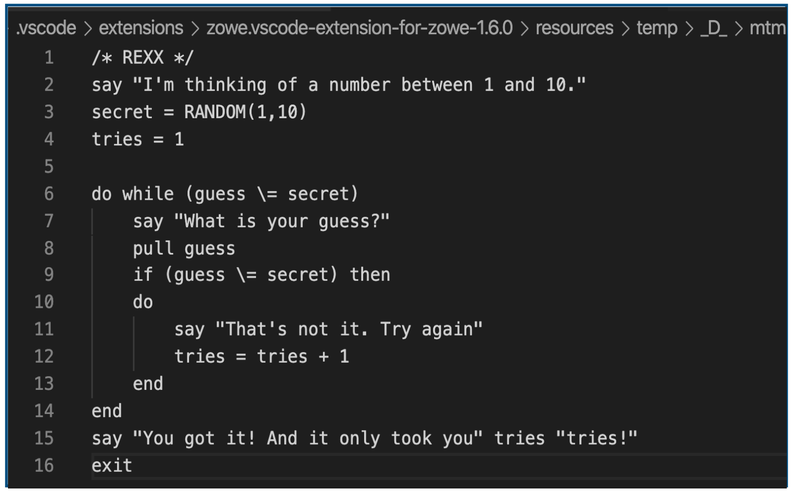
This is a program that generates a random number between 1 and 10, and you have to guess that number. There may be other games you’d rather be playing right about now, but not all of them will teach you about Rexx and TSO, so keep that in mind when looking over at your PS4 controller.



8. SEND YOUR GUESS

Send your guesses to the program by replacing everything between the double-quotes with a number. You can see here that we’re using the persistent address space key to ensure our input goes to the TSO address space, which is sitting there waiting for our input.

If you get it right on the first try, congratulations! Feel free to start the program again and make sure you can see it go through the "Try again" steps.



9. LOOK AT THE CODE

If you haven’t already, load up the code for that program in your VS Code editor. It’s not a complicated program by any means, but you might be noticing just how simple this Rexx code really is. 16 lines of code, including a comment and a blank, and yes, those “say” and “pull” commands really do what you think they do.

You can see why people love this language. It’s also got what I consider to be the world’s greatest logo for a programming language. Look at it. Just. Look. At. It.

“WHY DO WE NEED ANOTHER PROGRAMMING LANGUAGE?"

For the same reason a well-stocked toolbox has a hammer, screwdrivers of various sizes and shapes, tape measure, drill bits, and whatever else the owner might need. You could probably use a hammer to measure things, or the broad side of a tape measure to hammer in a nail, but there are tools built specifically to handle certain tasks best. There are times when a simple Bash shell script will do the trick, and other times where you want some highly-structured COBOL, Java or C++.

Rexx excels where you need to script or automate commands and actions. Your average system programmer will have a whole data set (or several) full of Rexx scripts they’ve collected over the years, and they get used to carry out work that would typically require a lot of hands-on-keyboard typing. There probably isn’t a single major company in the world running mainframe that doesn’t use it.

```
Monorail:~ jbisti$ zowe tso send as Z99999-102-aabaaaan --data "exec 'Z99999.source(guessnum
I'm thinking of a number between 1 and 10.
What is your guess?

Monorail:~ jbisti$ zowe tso send as Z99999-102-aabaaaan --data "11"
What part of 1-10 did you not understand?
That's not it. Try again
What is your guess?

Monorail:~ jbisti$ zowe tso send as Z99999-102-aabaaaan --data "3.7"
A WHOLE number. Duh.
That's not it. Try again
What is your guess?

Monorail:~ jbisti$ zowe tso send as Z99999-102-aabaaaan --data "Q"
That's not even a number, you silly goose!
That's not it. Try again
What is your guess?

Monorail:~ jbisti$ zowe tso send as Z99999-102-aabaaaan --data "2"
That's not it. Try again
What is your guess?

Monorail:~ jbisti$ zowe tso send as Z99999-102-aabaaaan --data "3"
That's not it. Try again
What is your guess?
```

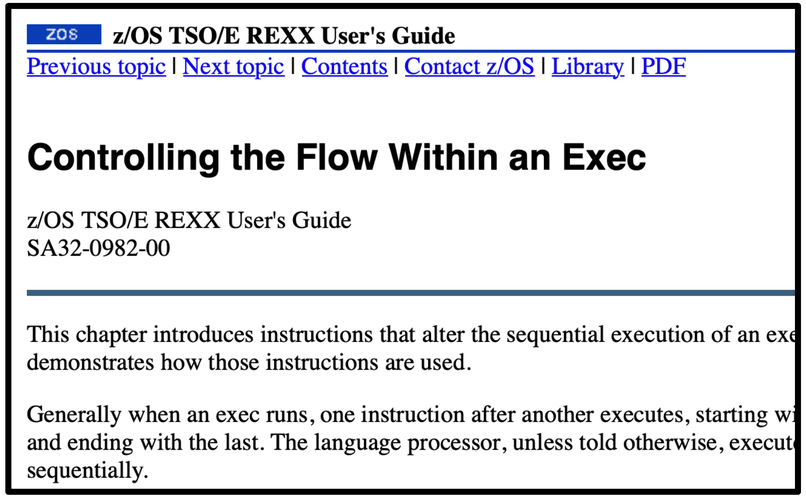
10. YELL AT THE USER

Now that we're in Part 3, we can be a little more creative with how we explore things. We want you to provide some feedback to the user if they do any of the following:

1) Enter a number less than 0
2) Enter a number greater than 10
3) Enter a number that is not an integer
4) Enter something that isn't even a number

Get creative, try stuff out. Feel free to be as nice or not-nice as you feel appropriate, but please keep profanity off of our system. We'd like to show off your work, and there are sensitive eyes out there.

While you're at it, give an option to quit the program. That way, you don't have to stop the address space to restart the program from the beginning.



z/OS **z/OS TSO/E REXX User's Guide**

Previous topic | Next topic | Contents | Contact z/OS | Library | PDF

Controlling the Flow Within an Exec

z/OS TSO/E REXX User's Guide
SA32-0982-00

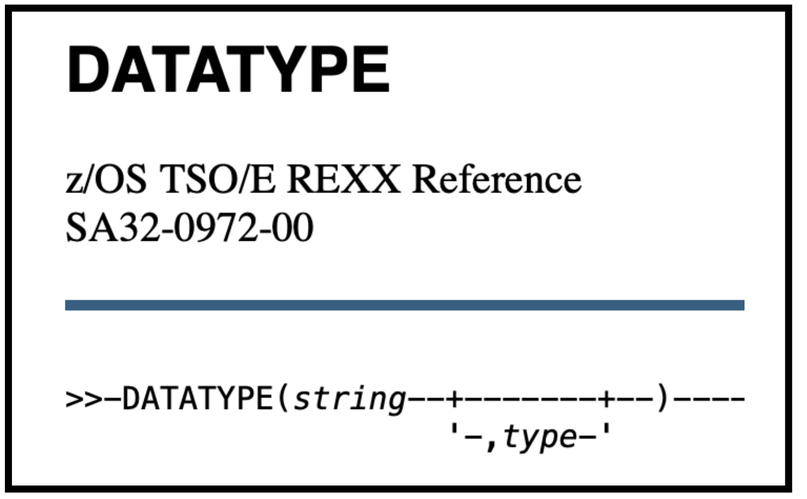
This chapter introduces instructions that alter the sequential execution of an exe
demonstrates how those instructions are used.

Generally when an exec runs, one instruction after another executes, starting wi
and ending with the last. The language processor, unless told otherwise, execut
sequentially.

11. GET LOOPY

We already coded in one IF/THEN statement, but you'll probably want to get a little more fancy. Visit the Knowledge Center for some good examples of controlling the program flow within an exec:

https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.ikjc300/loop.htm



DATATYPE

z/OS TSO/E REXX Reference
SA32-0972-00

```
>>-DATATYPE(string--+-------+--)----
                    '-,type-'
```

12. NOT MY TYPE

Make sure your finished product is called SOURCE(GUESSNUM) and that it does everything listed in Step 10. One last hint, check out the **datatype** function. It's very necessary. *VERY. NECESSARY.*

When you think you're done, test it with a few different conditions. Try to trick your logic and break the program. Once you've got it all locked down, submit your work and take a look at the next Rexx challenge.

NICE JOB! LET'S RECAP

Now you're getting into the swing of things. You're interacting with a Rexx program, running in a TSO address space, and you're doing that through the zowe command.

You've probably also learned quite a bit about the Rexx language, and may have even done some extra reading about Address Spaces. Everything in here will help you become a skilled mainframe professional.

NEXT UP...

The iron is most definitely smoldering, and you're probably becoming a fan of Rexx. In REXX2, you'll write your own code from scratch, and implement some file reads/writes. Make that your next stop.