COBOL Programming Course #2 3.2.0

Learning COBOL

#2 3.2.0| 250929-2138

Copyright Contributors to the Open Mainframe Project's COBOL Programming Course

1 WHY COBOL?

This chapter introduces COBOL, specifically regarding its use in enterprise systems.

- **What is COBOL?**
- **How is COBOL being used today?**
- **What insights does the survey conducted by Open Mainframe Project's COBOL Working Group provide?**
- **Why should I care about COBOL?**

1.1 WHAT IS COBOL?

One computer programming language was designed specifically for business, Common Business-Oriented Language, COBOL. COBOL has been transforming and supporting business globally since its invention in 1959. COBOL is responsible for the efficient, reliable, secure, and unseen day-to-day operation of the world's economy. The day-to-day logic used to process our most critical data is frequently done using COBOL.

Many COBOL programs have decades of improvements which include business logic, performance, programming paradigm, and application program interfaces to transaction processors, data sources, and the Internet.

Many hundreds of programming languages were developed during the past 60 years with expectations to transform the information technology landscape. Some of these languages, such as C, C++, Java, and JavaScript, have indeed transformed the ever-expanding information technology landscape. However, COBOL continues to distinguish itself from other programming languages due to its inherent ability to handle vast amounts of critical data stored in the largest servers such as the IBM Z mainframe.

Continuously updated to incorporate modernized and proven programming paradigms and best practices, COBOL will remain a critical programming language into the foreseeable future. Learning COBOL enables you to read and understand the day-to-day operation of critical systems. COBOL knowledge and proficiency is a required skill to be a "full-stack developer" in large enterprises.

Copyright Contributors to the Open Mainframe Project's COBOL Programming Course

1.2 HOW IS COBOL BEING USED TODAY?

COBOL is omnipresent, and it's highly likely that you've interacted with a COBOL application today. Let's take a look at some compelling statistics:

- Approximately 95% of ATM transactions rely on COBOL codes.
- COBOL powers 80% of face-to-face transactions.
- Each day, COBOL systems facilitate a staggering $3 trillion in commerce.

To truly grasp the extent of COBOL's prevalence, consider these mind-boggling facts:

- On a daily basis, there are 200 times more COBOL transactions executed than there are Google searches.
- Presently, there are over 250 billion lines of actively running COBOL programs, accounting for roughly 80% of the world's actively utilized code.
- Every year, approximately 1.5 billion lines of new COBOL code are written.

1.3 WHAT INSIGHTS DOES THE SURVEY CONDUCTED BY OPEN MAINFRAME PROJECT'S COBOL WORKING GROUP PROVIDE?

Highlights from the survey conducted by the Open Mainframe Project's COBOL Working Group in 2021 shed further light on the use of COBOL in today's era:

1. **Over 250 billion lines of COBOL are currently in production worldwide**, marking a notable increase from previous estimates. This figure indicates that COBOL's relevance is not waning; in fact, it continues to play a vital role.

   Industries with the greatest reliance on COBOL include Financial Services, Government, Software, Logistics, Retail, and Manufacturing, among others.

2. **The future of COBOL appears promising**. Despite skeptics, 58% of respondents anticipate that their COBOL applications will persist for at least the next five years. Financial services professionals, in particular, exhibit even greater optimism, with over 55% expecting COBOL to endure indefinitely.

Copyright Contributors to the Open Mainframe Project's COBOL Programming Course

3. **The challenge lies in the availability of COBOL skills**. Interestingly, COBOL was originally designed to be an accessible language that didn't necessitate advanced computing expertise.However, companies are worried about not having enough skilled COBOL programmers. Even though they can train their own employees, this concern shows that COBOL is still important and worth investing in.

1.4 WHY SHOULD I CARE ABOUT COBOL?

The COBOL programming language, COBOL compiler optimization, and COBOL run time performance have over 50 years of technology advancements that contribute to the foundation of the world's economy. The core business logic of many large enterprises has decades of business improvement and tuning embedded in COBOL programs.

The point is - whatever you read or hear about COBOL, be very skeptical. If you have the opportunity to work directly with someone involved in writing or maintaining critical business logic using COBOL, you will learn about the operation of the core business. Business managers, business analysts, and decision-makers come and go. The sum of all good business decisions can frequently be found in the decades of changes implemented in COBOL programs. The answer to "How does this business actually work?" can be found in COBOL programs.

COBOL Programming Course #2 3.2.0I
250929-2138

Add the following to your awareness of COBOL. It is an absolute myth that you must be at least 50 years old to be good with COBOL. COBOL is incredibly easy to learn and understand. One of the many reasons financial institutions like COBOL is the fact that it is not necessary to be a programmer to read and understand the logic. This is important because critical business logic code is subject to audit. Auditors are not programmers. However, auditors are responsible for ensuring the business financial statements are presented fairly. It is COBOL processing that frequently results in the business ledger updates and subsequent financial statements.

Now for a real-world lesson. A comment recently made in a well-known business journal by someone with a suspect agenda was quoted as saying, "COBOL is a computing language used in business and finance. It was first designed in 1959 and is pretty old and slow." A highly experienced business technology person knows the only true part of that last sentence was that COBOL was first designed in 1959.

It's no secret that lots of banks still run millions of lines of COBOL on mainframes. They probably want to replace that at some point. So why haven't they? Most banks have been around long enough to still feel the pain from the ~1960's software crisis. After spending enormous amounts of money, and time, on developing their computer systems, they finally ended up with a fully functional, well-tested, stable COBOL core system.

Speaking with people that have worked on such systems, nowadays they have Java front ends and wrappers which add functionality or more modern interfaces, they run the application on virtualized replicated servers, but in the end, everything runs through that single-core logic. And that core logic is rarely touched or changed, unless necessary.

From a software engineering perspective, that even makes sense. Rewrites are always more expensive than planned, and always take longer than planned (OK, probably not always. But often.). Never change a running system etc., unless very good technical and business reasons exist.

COBOL Programming Course #2 3.2.01 250929-2138

2 BASIC COBOL

This chapter introduces the basics of COBOL syntax. It then demonstrates how to view and run a basic COBOL program in VS Code.

- **COBOL characteristics**
  - **Enterprise COBOL**
  - **Chapter objectives**
- **What must a novice COBOL programmer know to be an experienced COBOL programmer?**
  - **What are the coding rules and the reference format?**
  - **What is the structure of COBOL?**
  - **What are COBOL reserved words?**
  - **What is a COBOL statement?**
  - **What is the meaning of a scope terminator?**
  - **What is a COBOL sentence?**
  - **What is a COBOL paragraph?**
  - **What is a COBOL section?**
  - **How to run a COBOL program on z/OS?**
- **COBOL Divisions**
  - **COBOL Divisions structure**
  - **What are the four Divisions of COBOL?**
- **PROCEDURE DIVISION explained**
- **Additional information**
  - **Professional manuals**
  - **Learn more about recent COBOL advancements**
- **Lab**

Copyright Contributors to the Open Mainframe Project's COBOL Programming Course

- **Lab - Zowe CLI & Automation**
    - **Zowe CLI - Interactive Usage**
    - **Zowe CLI - Programmatic Usage**

2.1 COBOL CHARACTERISTICS

COBOL is an English-like computer language enabling COBOL source code to be easier to read, understand, and maintain. Learning to program in COBOL includes knowledge of COBOL source code rules, COBOL reserved words, COBOL structure, and the ability to locate and interpret professional COBOL documentation. These COBOL characteristics must be understood to be proficient in reading, writing, and maintaining COBOL programs.

2.1.1 Enterprise COBOL

COBOL is a standard and not owned by any company or organization. “Enterprise COBOL” is the name for the COBOL programming language compiled and executed in the IBM Z Operating System, z/OS. COBOL details and explanations in the following chapters apply to Enterprise COBOL.

Enterprise COBOL has decades of advancements, including new functions, feature extensions, improved performance, application programming interfaces (APIs), etc. It works with modern infrastructure technologies with native support for JSON, XML, and Java®.

2.1.2 Chapter Objectives

The object of the chapter is to expose the reader to COBOL terminology, coding rules, and syntax while the remaining chapters include greater detail with labs for practicing what is introduced in this chapter.

Copyright Contributors to the Open Mainframe Project's COBOL Programming Course

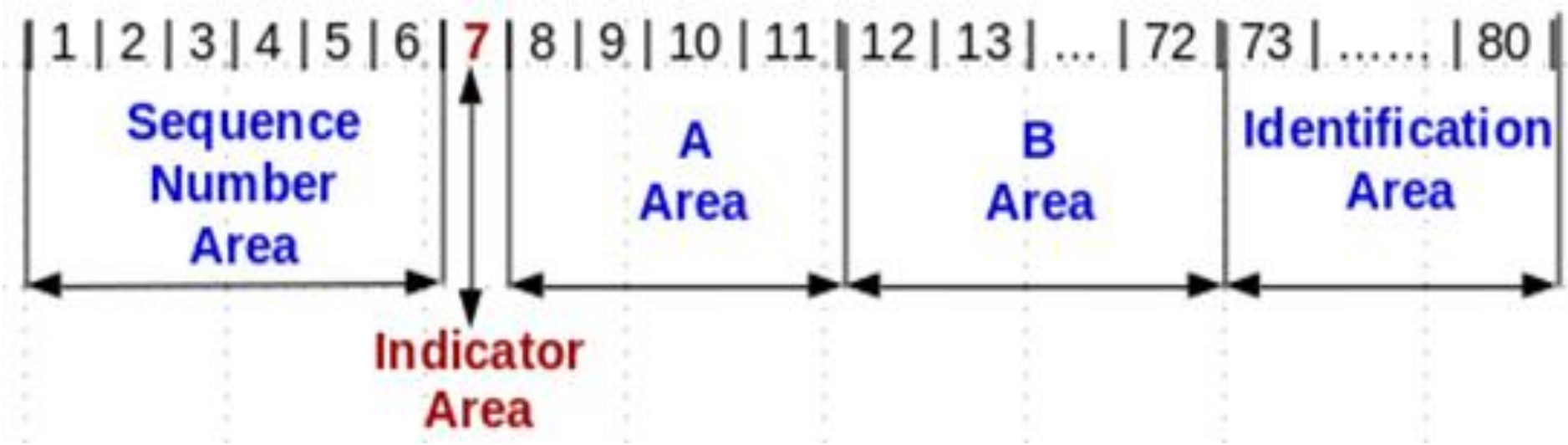
2.2 WHAT MUST A NOVICE COBOL PROGRAMMER KNOW TO BE AN EXPERIENCED COBOL PROGRAMMER?

This section will provide the reader with the information needed to more thoroughly understand the questions and answers being asked in each subsequent heading.

2.2.1 What Are The Coding Rules And The Reference Format?

COBOL source code is column-dependent, meaning column rules are strictly enforced. Each COBOL source code line has five areas, where each of these areas has a beginning and ending column.

COBOL source text must be written in COBOL reference format. Reference format consists of the areas depicted in Figure 1. in a 72-character line.



COBOL Programming Course #2 3.2.0|
250929-2138

Figure 1. COBOL reference format

The COBOL reference format is formatted as follows:

2.2.1.1 Sequence Number Area (Columns 1 - 6)

- Blank or reserved for line sequence numbers.

2.2.1.2 Indicator Area (Column 7)

- A multi-purpose area:
    - Comment line (generally an asterisk symbol)
    - Continuation line (generally a hyphen symbol)
    - Debugging line (D or d)
    - Source listing formatting (a slash symbol)

2.2.1.3 Area A (Columns 8 - 11)

- Certain items must begin in Area A, they are:
    - Level indicators
    - Declarative
    - Division, Section, Paragraph headers
    - Paragraph names
- Column 8 is referred to as the A Margin

2.2.1.4 Area B (Columns 12 - 72)

- Certain items must begin in Area B, they are:
    - Entries, sentences, statements, and clauses

COBOL Programming Course #2 3.2.01 250929-2138

    - Continuation lines

- Column 12 is referred to as the B Margin

2.2.1.5 Identification Area (Columns 73 - 80)

- Ignored by the compiler.
- Can be blank or optionally used by the programmer for any purpose.

2.2.2 What Is The Structure Of COBOL?

COBOL is a hierarchy structure consisting and in the top-down order of:

- Divisions
- Sections
- Paragraphs
- Sentences
- Statements

2.2.3 What Are COBOL Reserved Words?

COBOL programming language has many words with specific meaning to the COBOL compiler, referred to as reserved words. These reserved words cannot be used as programmer chosen variable names or programmer chosen data type names.

A few COBOL reserved words pertinent to this book are: PERFORM, MOVE, COMPUTE, IF, THEN, ELSE, EVALUATE, PICTURE, etc. You can find a table of all COBOL reserved words is located at:

https://www.ibm.com/docs/en/cobol-zos/6.4?topic=appendixes-reserved-words

COBOL Programming Course #2 3.2.01 250929-2138

2.2.4 What Is A COBOL Statement?

Specific COBOL reserved words are used to change the execution flow based upon current conditions. “Statements” only exist within the Procedure Division, the program processing logic. Examples of COBOL reserved words used to change the execution flow are:

- IF
- Evaluate
- Perform

2.2.5 What Is The Meaning Of A Scope Terminator?

A scope terminator can be explicit or implicit. An explicit scope terminator marks the end of certain PROCEDURE DIVISION statements with the “END-” COBOL reserved word. Any COBOL verb that is either, always conditional (IF, EVALUATE), or has a conditional clause (COMPUTE, PERFORM, READ) will have a matching scope terminator. An implicit scope terminator is a period (.) that ends the scope of all previous statements that have not yet been ended.

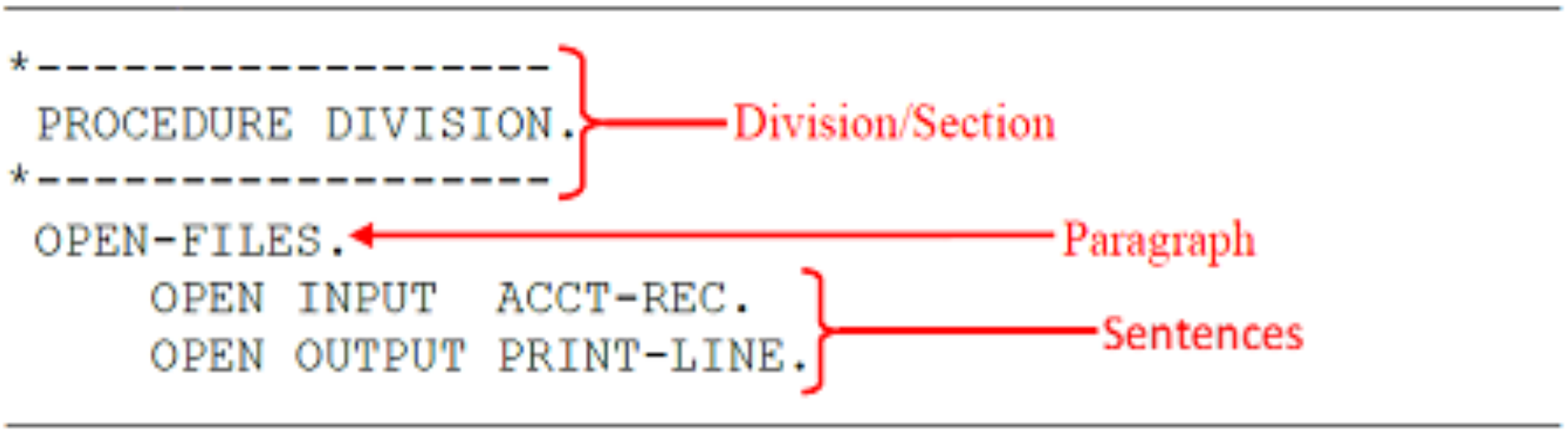
2.2.6 What Is A COBOL Sentence?

A COBOL “Sentence” is one or more “Statements” followed by a period (.), where the period serves as a scope terminator.

2.2.7 What Is A COBOL Paragraph?

A COBOL “Paragraph” is a user-defined or predefined name followed by a period. A “Paragraph” consists of zero or more sentences and is the subdivision of a “Section” or “Division”, see Example 1. below.

Copyright Contributors to the Open Mainframe Project's COBOL Programming Course



Example 1. Division -> paragraph -> sentences

2.2.8 What Is A COBOL Section?

A “Section” is either a user-defined or a predefined name followed by a period and consists of zero or more sentences. A “Section” is a collection of paragraphs.

2.2.9 How To Run A COBOL Program On Z/OS?

When you are dealing with COBOL on z/OS, you will encounter JCL or Job Control Language. JCL is a set of statements that tell the z/OS operating system about the tasks you want it to perform.

For your COBOL program to be executable in z/OS, you will need to tell the operating system to compile and link-edit the code before running it. All of which will be done using JCL.

The first thing your JCL should do is compile the COBOL program you have written. In this step, your program is passed to the COBOL compiler to be processed into object code. Next, the output from the compiler will go through the link-edit step. Here a binder will take in the object code and all the necessary libraries and options specified in the JCL to create an executable program. In this step, you can also tell the JCL to include additional data sets which your COBOL program will read. Then, you can run the program.

COBOL Programming Course #2 3.2.0I 250929-2138

To simplify things, Enterprise COBOL for z/OS provides three JCL procedures to compile your code. When using a JCL procedure, we can supply the variable part to cater to a specific use case. Listed below are the procedures available to you:

1. Compile procedure (IGYWC)
2. Compile and link-edit procedure (IGYWCL)
3. Compile, link-edit, and run procedure (IGYWCLG)

Since this course is a COBOL course, the JCL necessary for you to do the Labs is provided for you. Therefore, you will encounter the procedures listed above on the JCL. If you want to create a new COBOL program, you can copy one of the JCL provided and modify it accordingly.

To read more on JCL, visit the IBM Knowledge Center:

https://www.ibm.com/docs/en/zos-basic-skills?topic=collection-basic-jcl-concepts

2.3 COBOL DIVISIONS

This section introduces the four COBOL Divisions and briefly describes their purpose and characteristics.

2.3.1 COBOL Divisions Structure

Divisions are subdivided into Sections.

Sections are subdivided into Paragraphs.

Paragraphs are subdivided into Sentences.

Sentences consist of Statements.

Statements begin with COBOL reserved words and can be subdivided into “Phrases”

2.3.2 What Are The Four Divisions Of COBOL?

2.3.2.1 IDENTIFICATION DIVISION

The IDENTIFICATION DIVISION identifies the program with a name and, optionally, gives other identifying information, such as the Author name, program compiled date (last modified), etc.

2.3.2.2 ENVIRONMENT DIVISION

The ENVIRONMENT DIVISION describes the aspects of your program that depend on the computing environment, such as the computer configuration and the computer inputs and outputs.

2.3.2.3 DATA DIVISION

The DATA DIVISION is where characteristics of data are defined in one of the following sections:

- FILE SECTION:

  Defines data used in input-output operations.

- LINKAGE SECTION:

  Describes data from another program. When defining data developed for internal processing.

- WORKING-STORAGE SECTION:

  Storage allocated and remaining for the life of the program.

- LOCAL-STORAGE SECTION:

  Storage is allocated each time a program is called and de-allocated when the program ends.

COBOL Programming Course #2 3.2.0|
250929-2138

2.3.2.4 PROCEDURE DIVISION

The PROCEDURE DIVISION contains instructions related to the manipulation of data and interfaces with other procedures are specified.

2.4 PROCEDURE DIVISION EXPLAINED

The PROCEDURE DIVISION is where the work gets done in the program. Statements are in the PROCEDURE DIVISION where they are actions to be taken by the program. The PROCEDURE DIVISION is required for data to be processed by the program. PROCEDURE DIVISION of a program is divided into sections and paragraphs, which contain sentences and statements, as described here:

- **Section** - A logical subdivision of your processing logic. A section has a header and is optionally followed by one or more paragraphs. A section can be the subject of a PERFORM statement. One type of section is for declarative. Declarative is a set of one or more special-purpose sections. Special purpose sections are exactly what they sound like, sections written for special purposes and may contain things like the description of inputs and outputs. They are written at the beginning of the PROCEDURE DIVISION, the first of which is preceded by the keyword DECLARATIVES and the last of which is followed by the keyword END DECLARATIVES.
- **Paragraph** - A subdivision of a section, procedure, or program. A paragraph can be the subject of a statement.
- **Sentence** - A series of one or more COBOL statements ending with a period.
- **Statement** - An action to be taken by the program, such as adding two numbers.
- **Phrase** - A small part of a statement (i.e. subdivision), analogous to an English adjective or preposition

COBOL Programming Course #2 3.2.01 250929-2138

2.5 ADDITIONAL INFORMATION

This section provides useful resources in the form of manuals and videos to assist in learning more about the basics of COBOL.

2.5.1 Professional Manuals

As Enterprise COBOL experience advances, the need for professional documentation is greater. An internet search for Enterprise COBOL manuals includes: “Enterprise COBOL for z/OS documentation library - IBM”, link provided below. The site content has tabs for each COBOL release level. As of December 2022, the current release of Enterprise COBOL is V6.4. Highlight the V6.4 tab, then select product documentation.

https://www.ibm.com/support/pages/enterprise-cobol-zos-documentation-library

Three ’Enterprise COBOL for z/OS” manuals are referenced throughout the chapters as sources of additional information, for reference and to advance the level of knowledge. They are:

1. Language Reference - Describes the COBOL language such as program structure, reserved words, etc.

   https://publibfp.dhe.ibm.com/epubs/pdf/igy6lr40.pdf

2. Programming Guide - Describes advanced topics such as COBOL compiler options, program performance optimization, handling errors, etc.

   https://publibfp.dhe.ibm.com/epubs/pdf/igy6pg40.pdf

3. Messages and Codes - To better understand certain COBOL compiler messages and return codes to diagnose problems.

   https://publibfp.dhe.ibm.com/epubs/pdf/c2746482.pdf

2.5.2 Learn More About Recent COBOL Advancements

- What’s New in Enterprise COBOL for z/OS V6.1:

  https://www.ibm.com/support/pages/cobol-v61-was-announced-whats-new

- What’s New in Enterprise COBOL for z/OS V6.2:

  https://www.ibm.com/support/pages/cobol-v62-was-announced-whats-new

- What’s New in Enterprise COBOL for z/OS V6.3:

https://www.ibm.com/support/pages/cobol-v63-was-announced-whats-new

- What's New in Enterprise COBOL for z/OS V6.4:

https://www.ibm.com/docs/en/cobol-zos/6.4?topic=wn-what-is-new-in-enterprise-cobol-zos-64-cobol-64-ptfs-installed

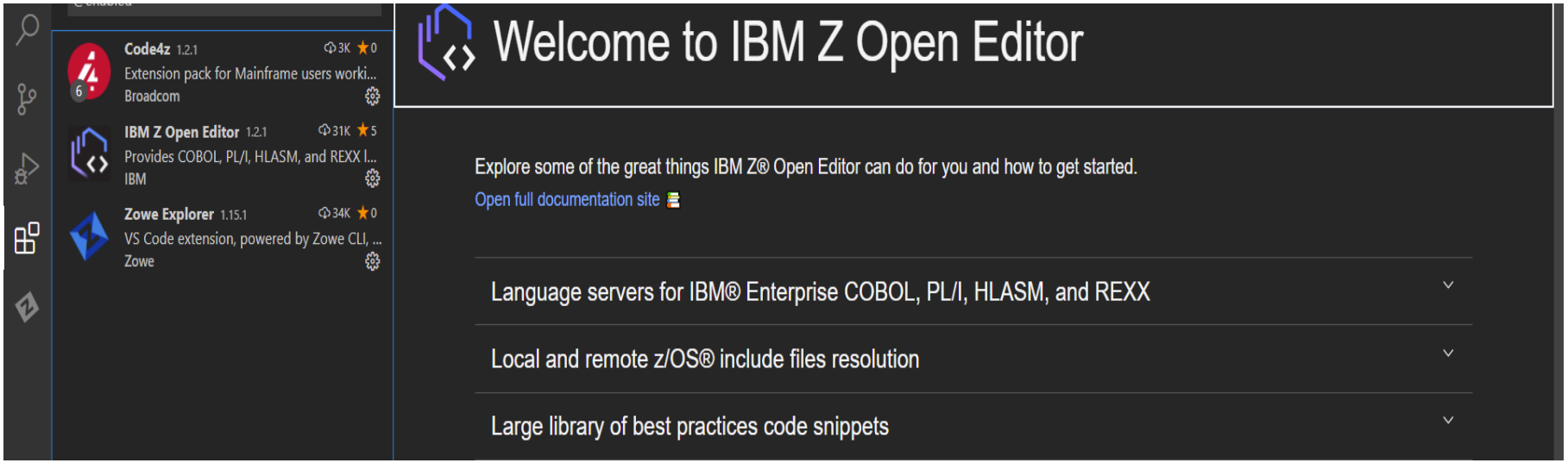
2.6 LAB

In this lab exercise, you will connect to an IBM Z system, view a simple COBOL hello world program in VS Code, submit JCL to compile the COBOL program, and view the output. Refer to "Installation of VS Code and extensions" to configure VS Code if you have not already done so. You can either use IBM Z Open Editor and Zowe Explorer, or Code4z.

1. The lab assumes installation of VS Code with either IBM Z Open Editor and Zowe Explorer extensions, as shown in Figure 2a, or the Code4z extension pack, as shown in Figure 2b.

   Click the **Extensions** icon. If you installed IBM Z Open Editor and Zowe Explorer, the list should include:

   1. IBM Z Open Editor
   2. Zowe Explorer



COBOL Programming Course #2 3.2.0|
250929-2138

Figure 2a. The IBM Z Open Editor and Zowe Explorer VS Code extensions

If you installed Code4z, the list should include:

1. COBOL Language Support
2. Zowe Explorer
3. Explorer for Endevor
4. HLASM Language Support
5. Debugger for Mainframe
6. COBOL Control Flow
7. Abend Analyzer for Mainframe
8. Data Editor for Mainframe

In these exercises, you will only use the COBOL Language Support and Zowe Explorer extensions.

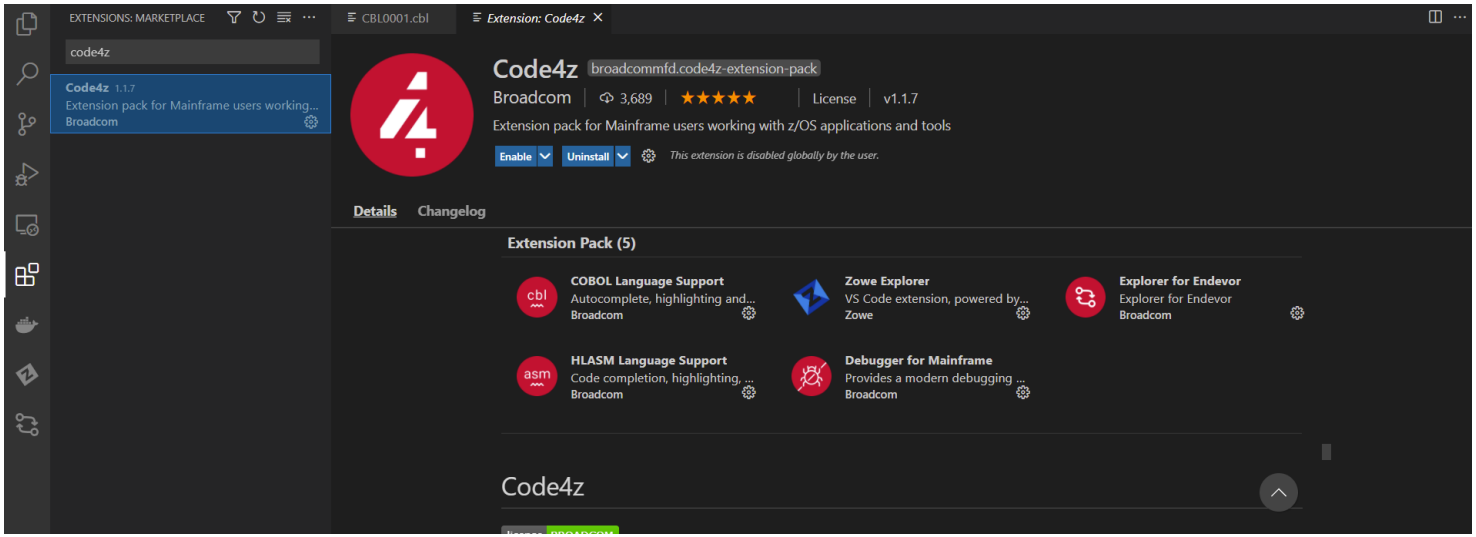


Figure 2b. The Code4z package of VS Code extensions.

Note: If your list contains both Z Open Editor and COBOL Language Support, disable one of them, by clicking on the **cog** icon next to the extension in the extensions list, and selecting **disable**.

2. Click the Zowe Explorer icon as shown in Figure 3. Zowe Explorer can list Data Sets, Unix System Services (USS) files, and Jobs output.

COBOL Programming Course #2 3.2.0|250929-2138

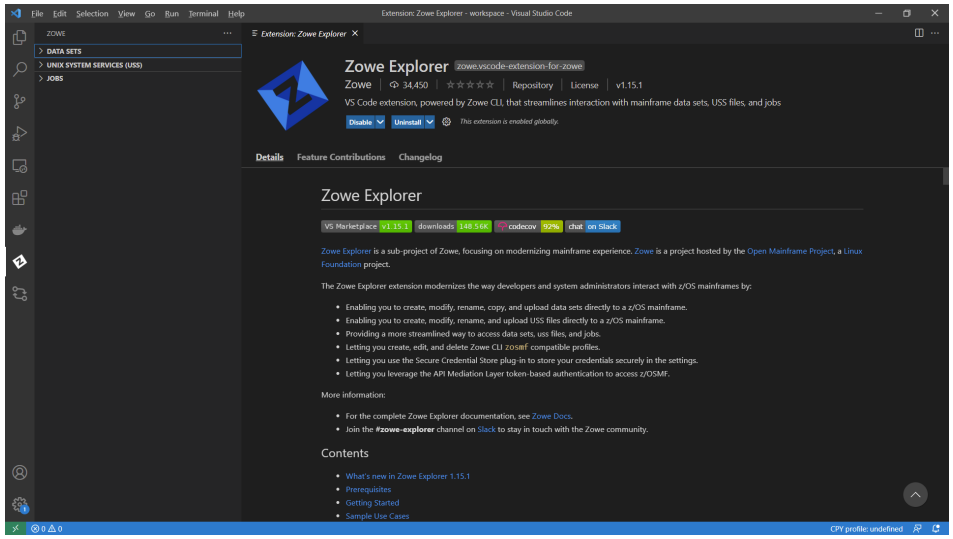


Figure 3. Zowe Explorer icon

3. In order to connect to the lab system, get your team configuration zip file and extract it. You can obtain the team configuration zip file from the Releases section of the course's GitHub repository.

Figure 4. Extract the ZIP file

4. Open the extracted folder. You will find the two configuration files as shown in Figure 5.

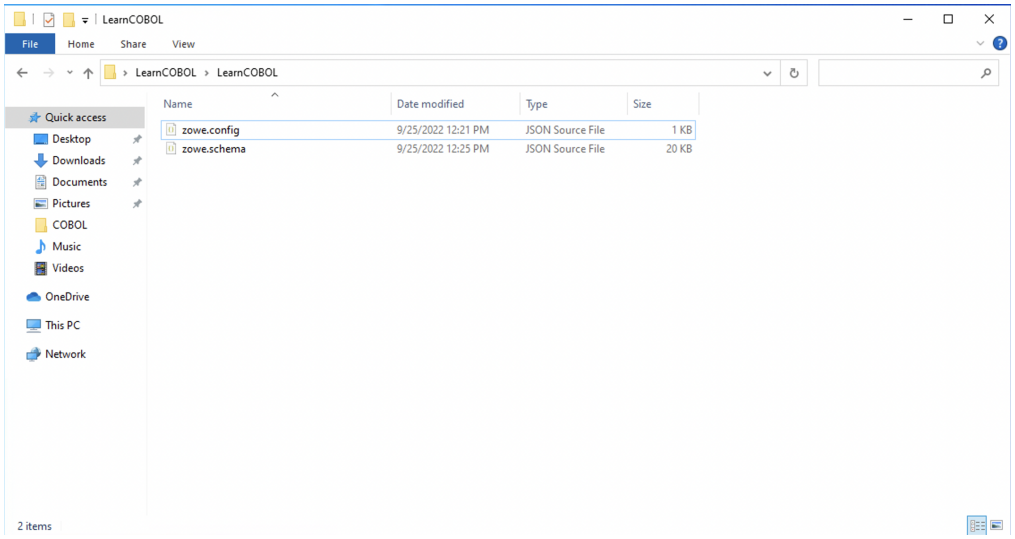


Figure 5. Inside the Team configuration file

5. Now back on your VS Code window, select the Explorer tab, and press the “Open folder” button in the left bar.

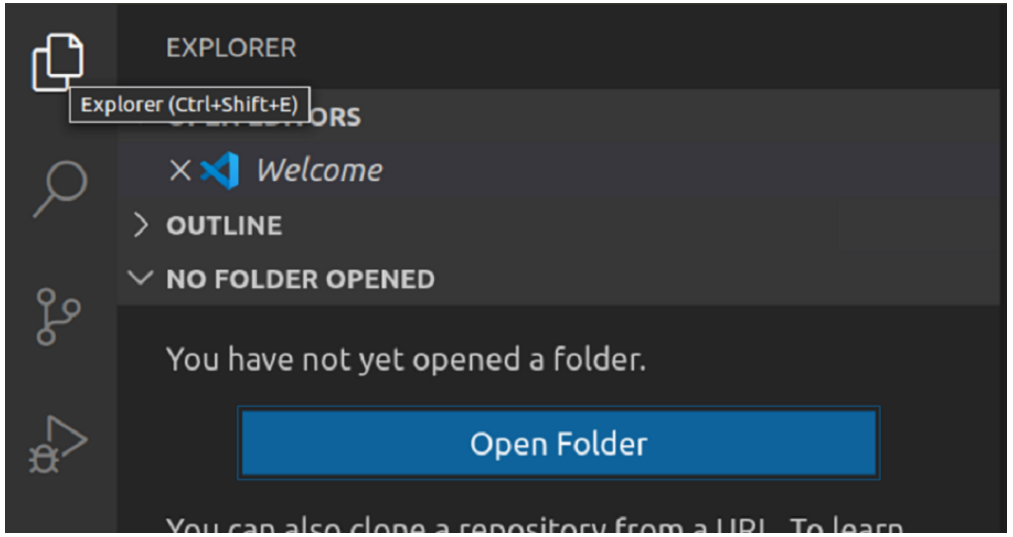


Figure 6. Click the open folder button

6. A pop-up window would show up, select the folder containing the team configuration files.

Copyright Contributors to the Open Mainframe Project's COBOL Programming Course

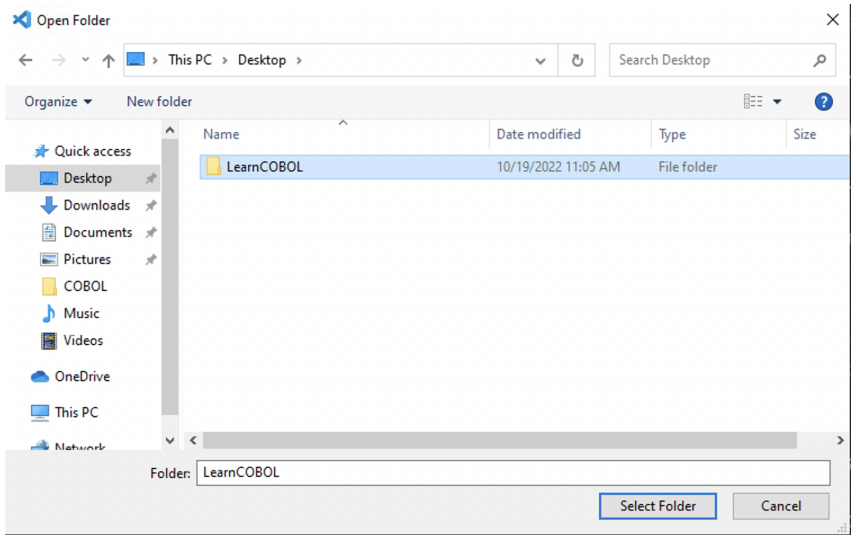


Figure 7a. Select the team Configuration folder

If you are prompted to trust the authors of the files in the folder as shown in Figure 7b, select **Yes, I trust the authors**.

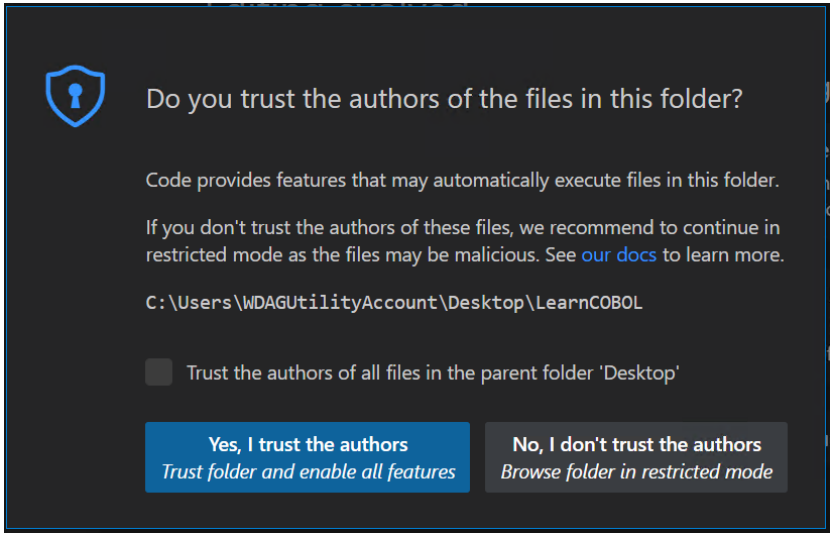


Figure 7b. Trust the authors of files the folder

7. Your connection should be added automatically to the Data Sets list as shown in Figure 8a.

COBOL Programming Course #2 3.2.0|
250929-2138

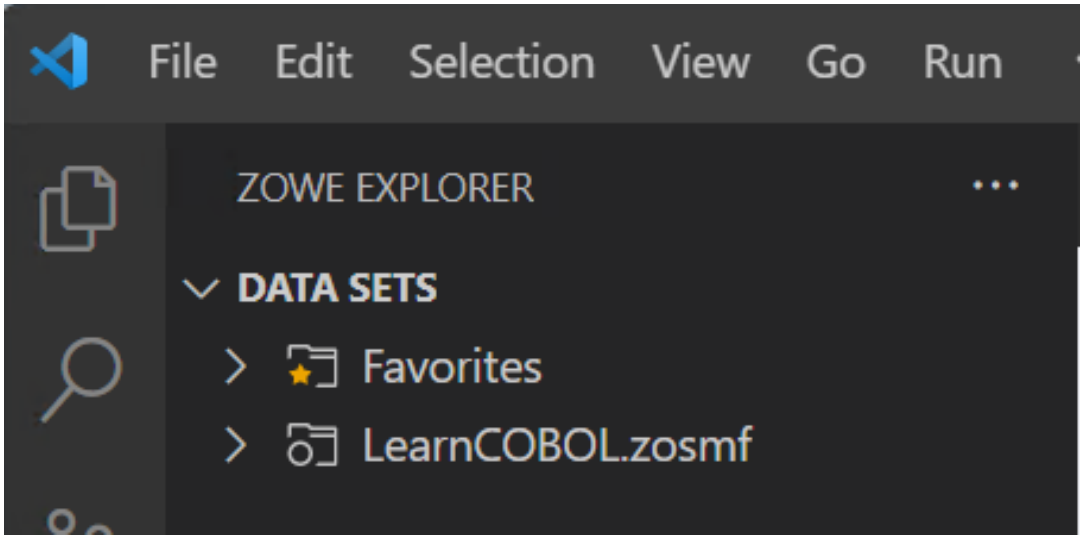


Figure 8a. LearnCOBOL Connection

If the connection does not appear, hover to the far right of the Data Sets line and press the + icon. Afterward, select the **LearnCOBOL** connection as shown in Figure 8b.

COBOL Programming Course #2 3.2.0|
250929-2138

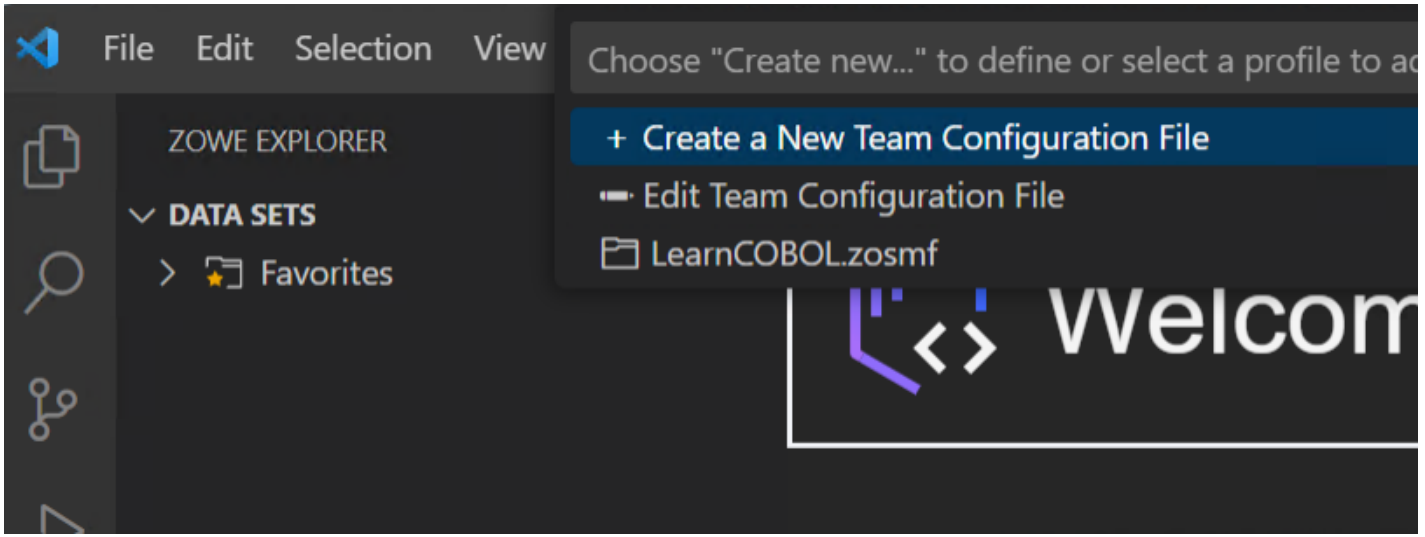


Figure 8b. Adding LearnCOBOL Connection manually

8. Press the LearnCOBOL connection.

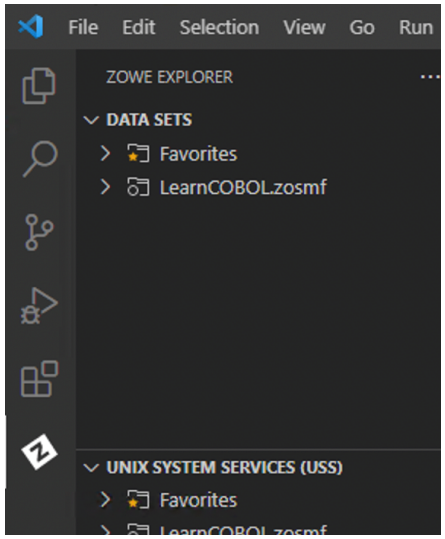


Figure 9. Pressing the LearnCOBOL Connection

9. The connection prompts for a username as shown in Figure 10.

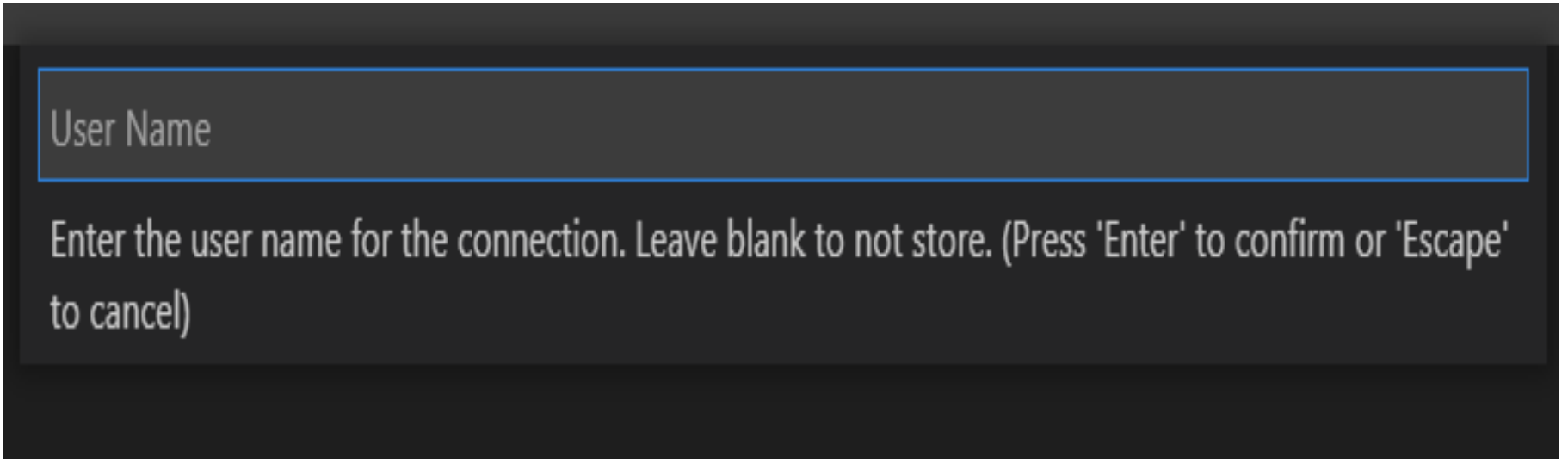


Figure 10. User name prompt

10. **Please enter the username assigned to you! Do not use the sample username of Z99998.** A sample username, is entered as shown in Figure 11. The ID is assigned by the System Administrator.

COBOL Programming Course #2 3.2.0|
250929-2138

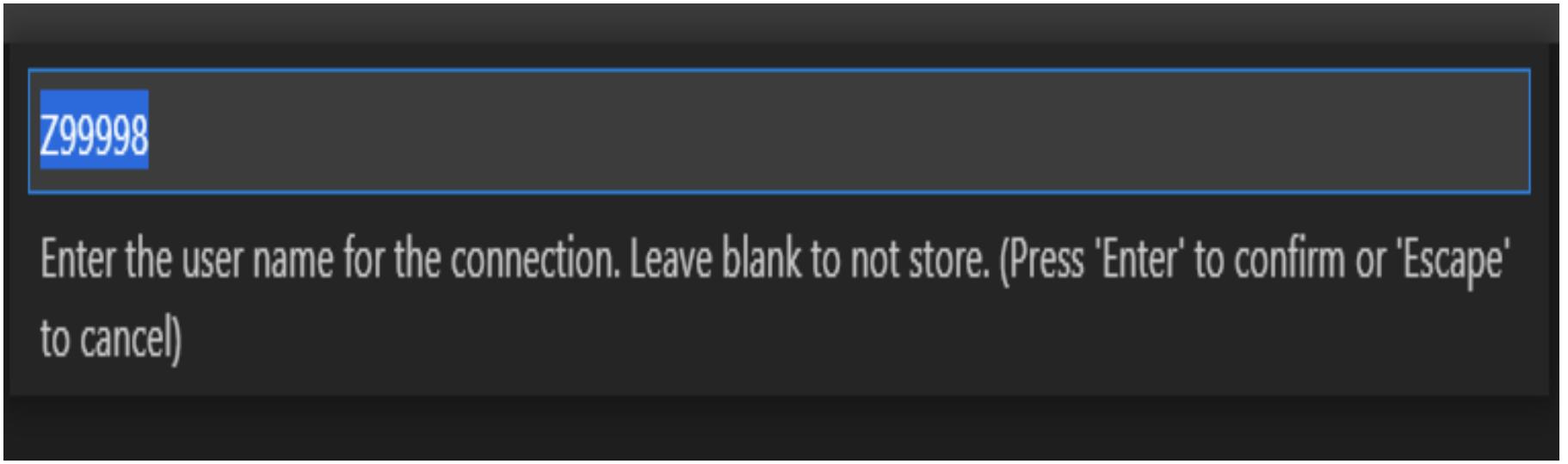


Figure 11. Specified user name

11. The connection prompts for a password as shown in Figure 12.

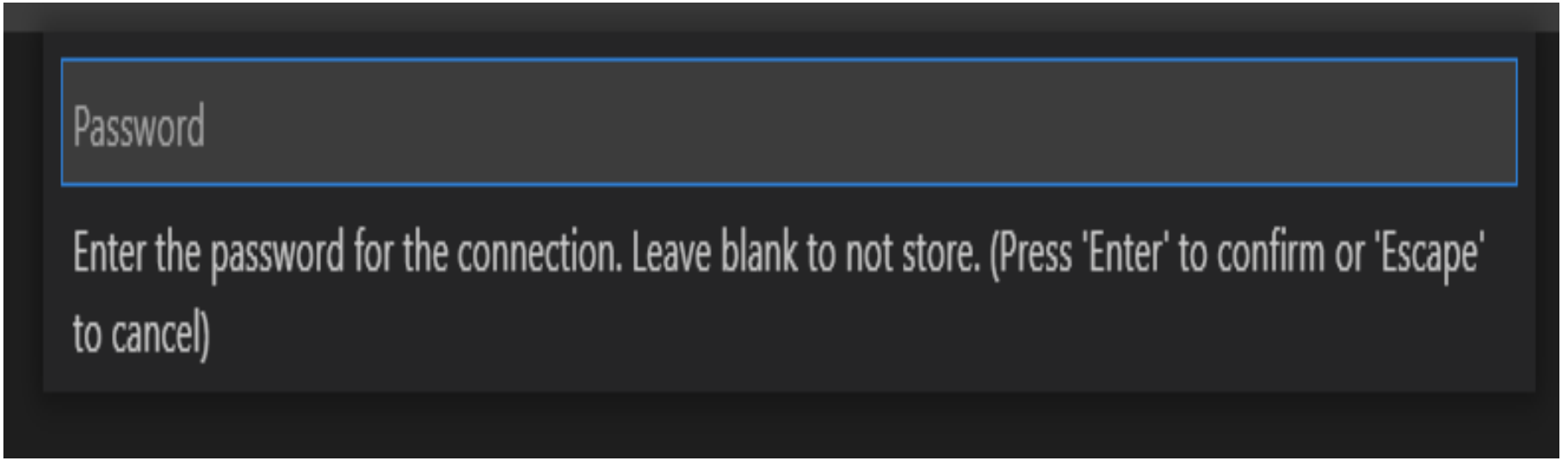


Figure 12. Password prompt

12. Enter your assigned password as shown in Figure 13.

COBOL Programming Course #2 3.2.0|
250929-2138

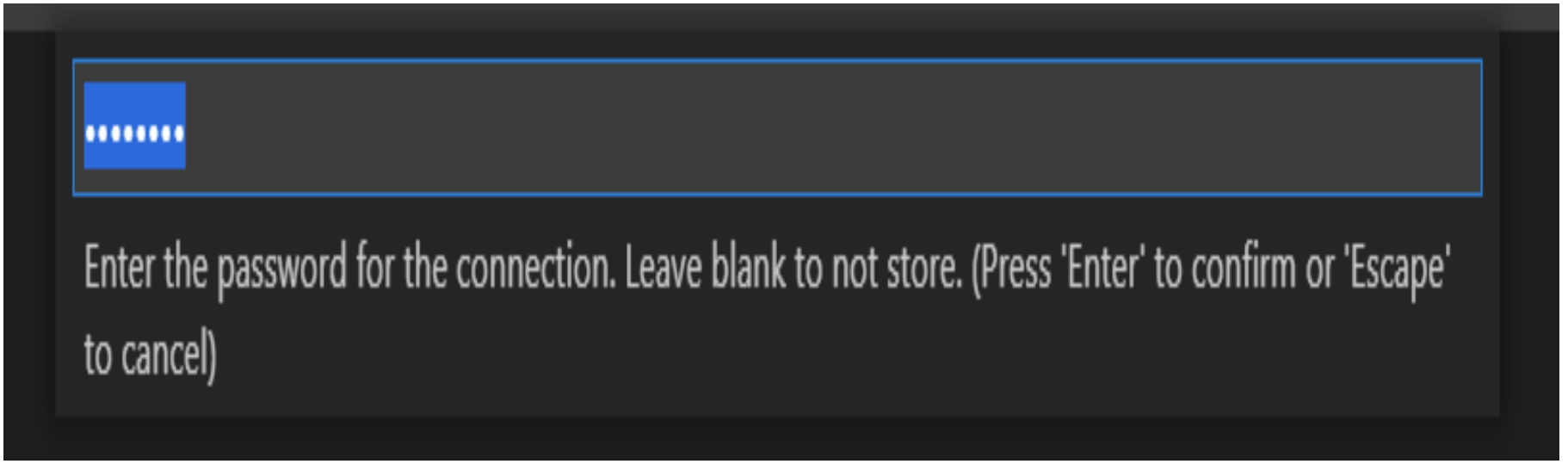


Figure 13. Specified password

13. Expanding LearnCOBOL shows “Use the search button to display datasets”. Click the magnifying glass icon as shown in Figure 14.

COBOL Programming Course #2 3.2.0|
250929-2138

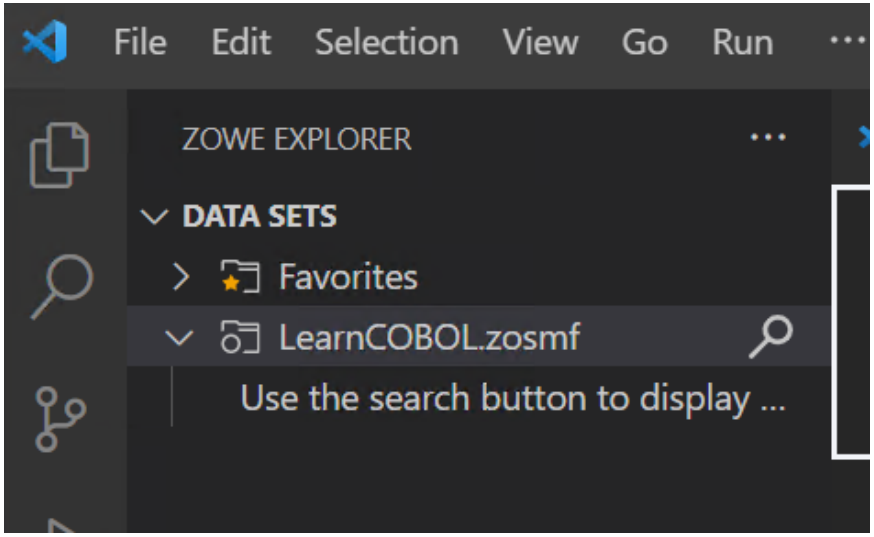


Figure 14. Magnifying glass icon to set a filter

14. A prompt to “Search Data Sets” will appear as shown in Figure 15.

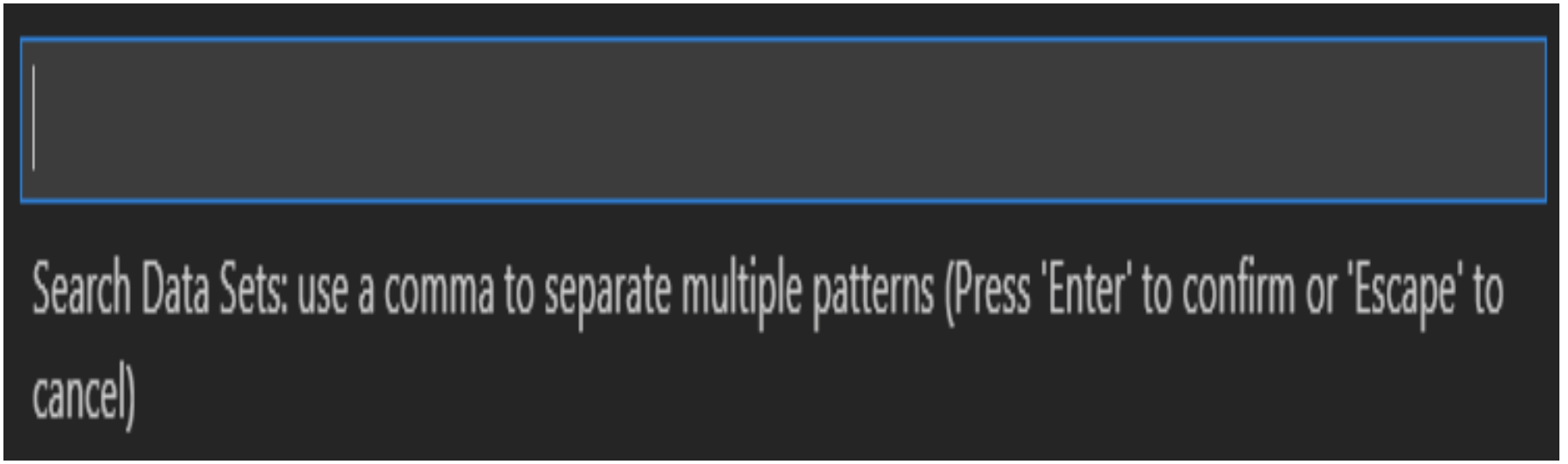


Figure 15. Filter name to be searched

15. Each user has a high-level qualifier that is the same as their username. Therefore, enter your assigned username as the search criteria as shown in Figure 16. **Please enter the username assigned to you! Do not use the sample username of Z99998.**

COBOL Programming Course #2 3.2.0|
250929-2138

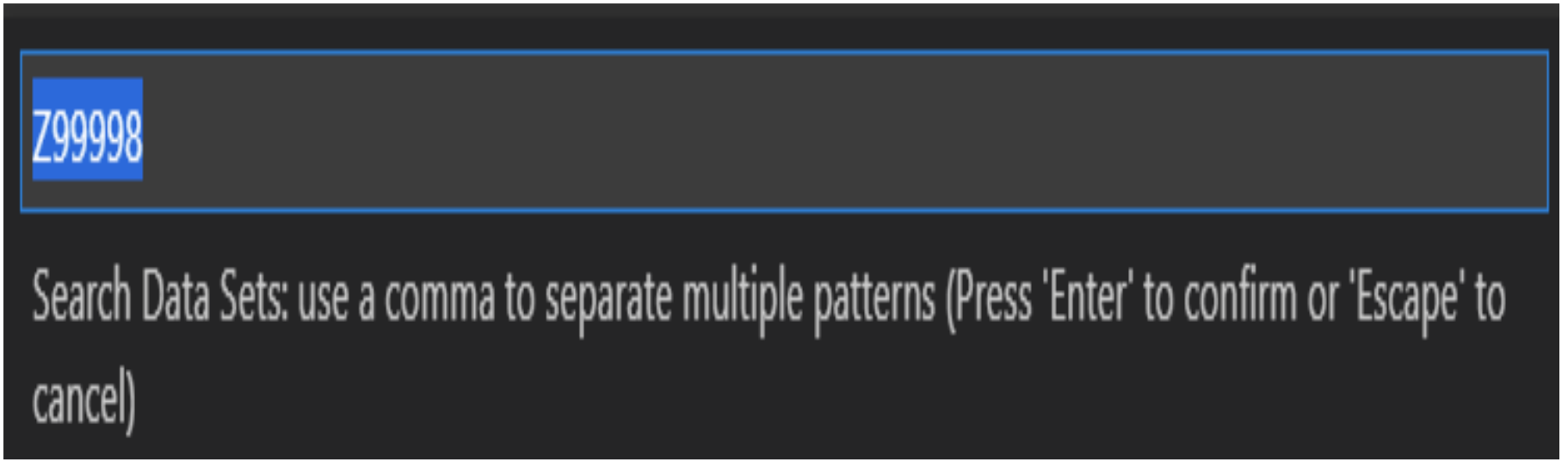


Figure 16. Entered filter name

16. A list of data set names beginning with your username will appear as shown in Figure 17.

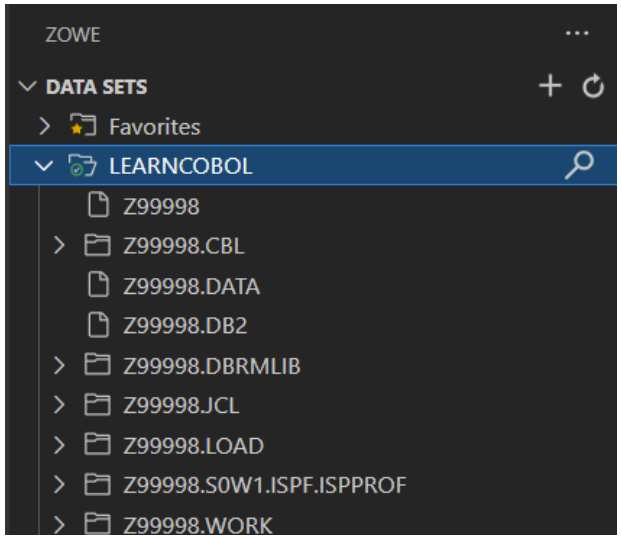


Figure 17. Filtered data set names

17. Expand **<USERNAME>.CBL** to view COBOL source members, then select member **HELLO** to see a simple COBOL 'Hello World!' program as shown in Figure 18.

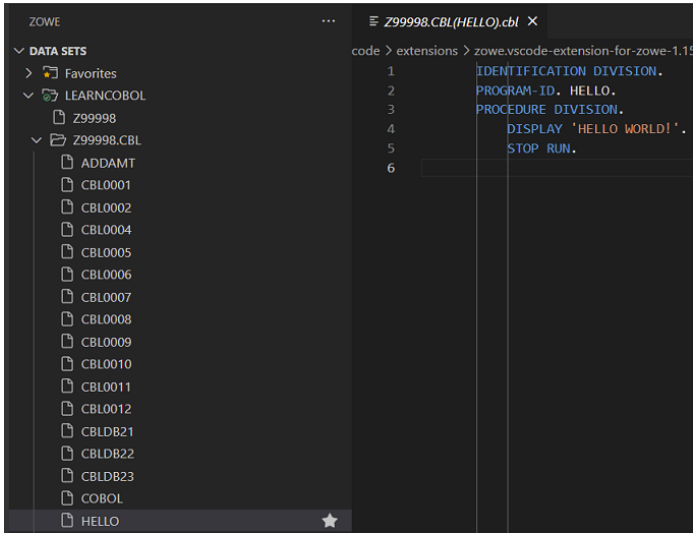


Figure 18. <USERNAME>.CBL

Copyright Contributors to the Open Mainframe Project's COBOL Programming Course

18. Expand **<USERNAME>.JCL** to view JCL members and select member HELLO which is the JCL used to compile and execute a simple 'Hello World!' COBOL source code as shown in Figure 19.

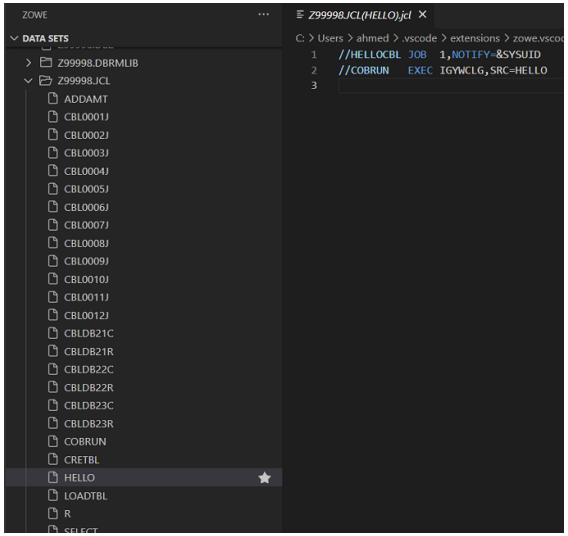
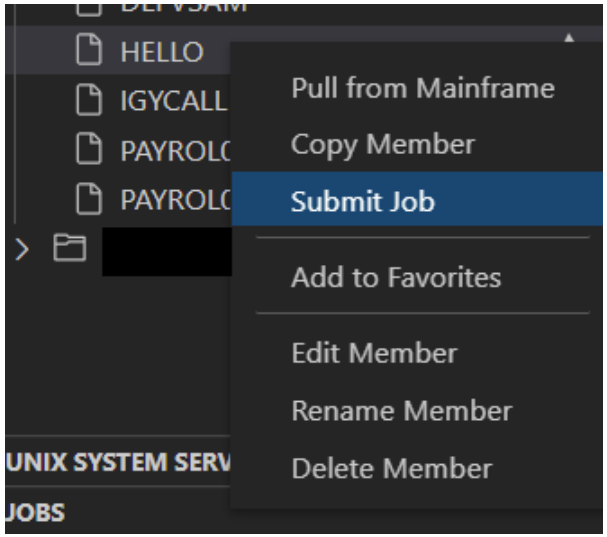


Figure 19. *<USERNAME>.JCL*

19. Right-click on JCL member **HELLO**. A section box appears. Select **Submit Job** for the system to process HELLO JCL as shown in Figure 20. The submitted JCL job compiles the COBOL HELLO source code, then executes the COBOL HELLO program.



COBOL Programming Course #2 3.2.0|
250929-2138

Figure 20. Submit Job

20. Observe the ‘Jobs’ section in Zowe Explorer as shown in Figure 21.

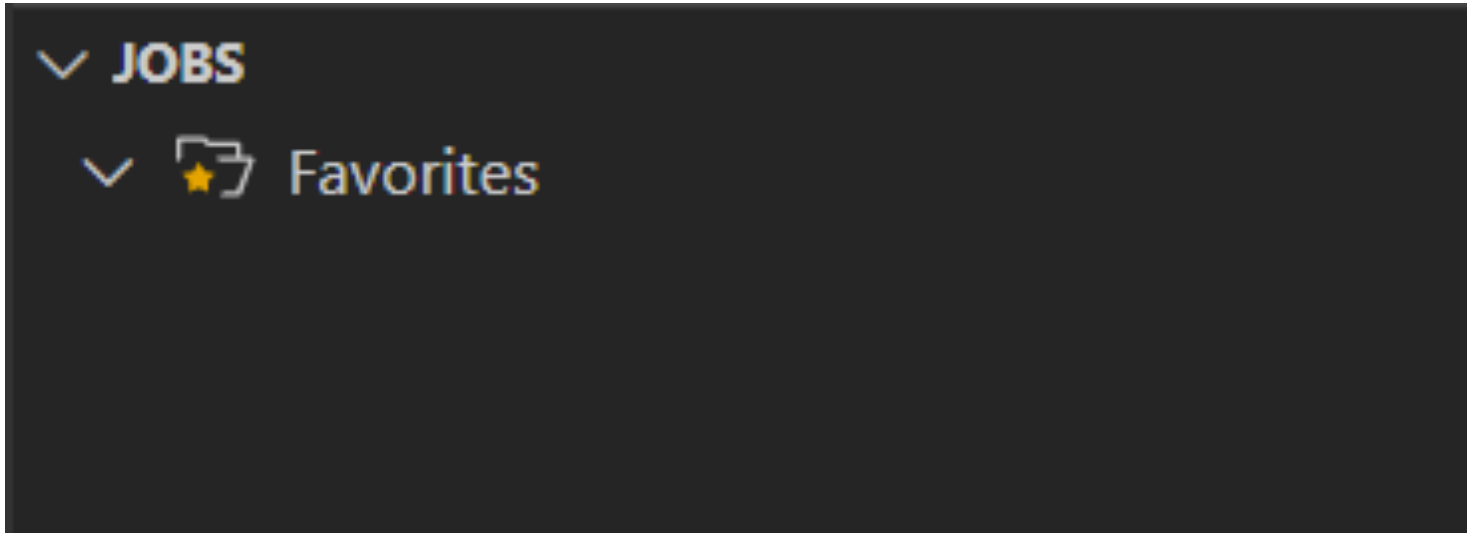


Figure 21. JOBS section

COBOL Programming Course #2 3.2.0|
250929-2138

21. Again, click on the + to the far right on the Jobs selection. The result is another prompt to ‘Create new’. Select **LearnCOBOL** from the list as shown in Figure 22.

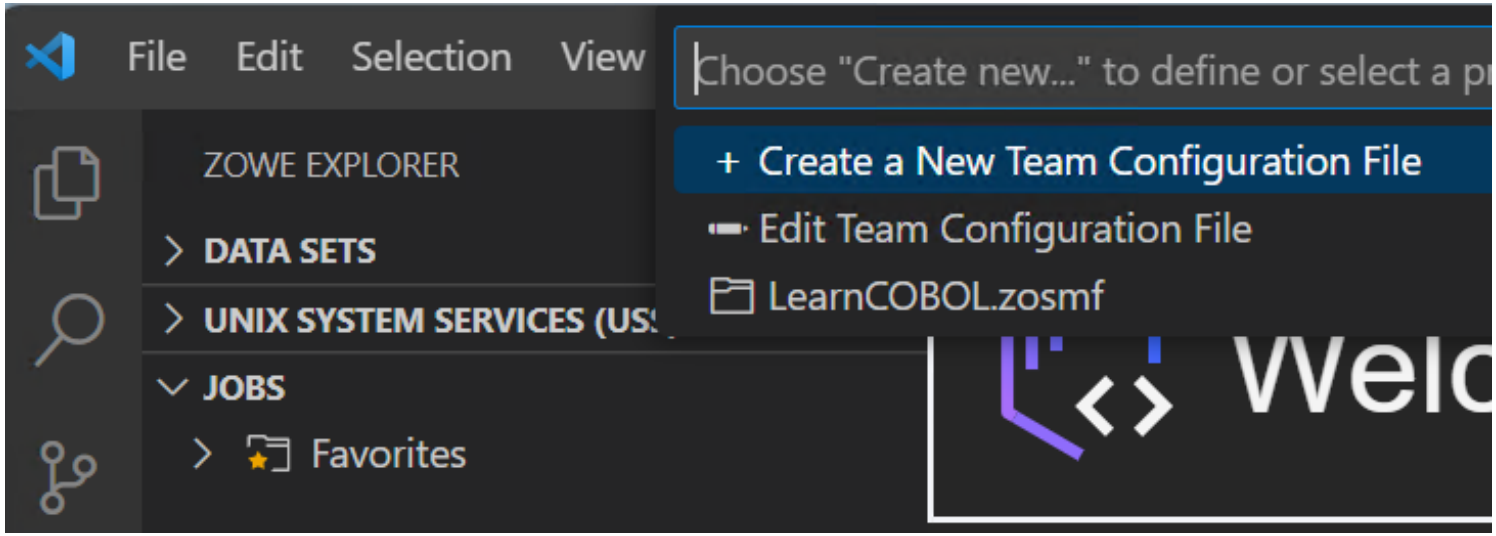


Figure 22. Select LearnCOBOL connection

22. As a result, the JCL jobs owned by your username appear. HELLOCBL is the JCL job name previously submitted. Expand **HELLOCBL** output to view sections of the output as shown in Figure 23.

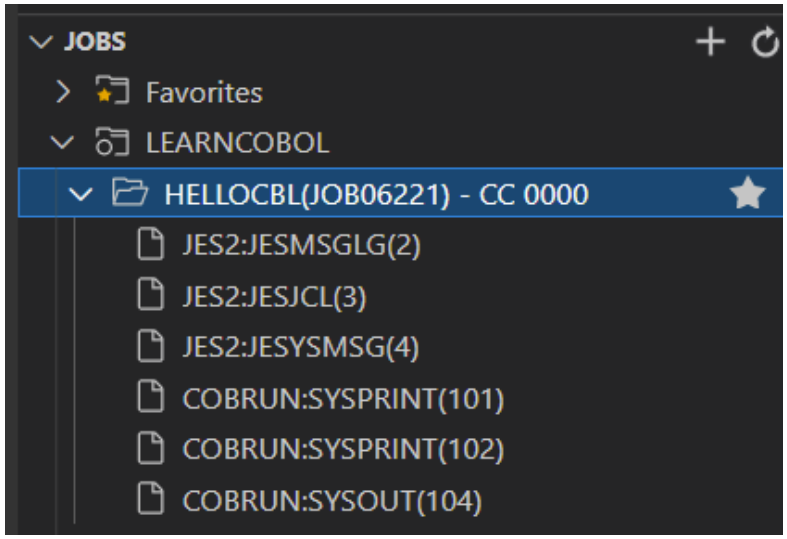


Figure 23. HELLOCBL output

23. Select **COBRUN:SYSPRINT(101)** to view the COBOL compiler output. Scroll forward in the COBOL compile to locate the COBOL source code compiled into an executable module as shown in Figure 24. Observe the Indicator Area in column 7, A Area beginning in column 8, and B Area beginning in column 12. Also, observe the period (.) scope terminators in the COBOL source.

COBOL Programming Course #2 3.2.0|
250929-2138

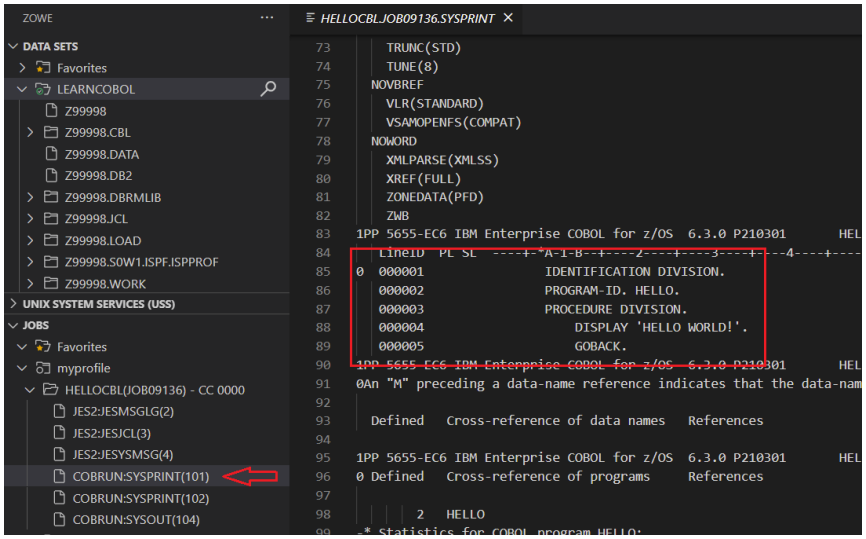


Figure 24. COBOL compiler output

24. View the COBOL program execution by selecting **COBRUN:SYSOUT(104)** from the LEARNCOBOL in the Jobs section of Zowe Explorer as shown in Figure 25.

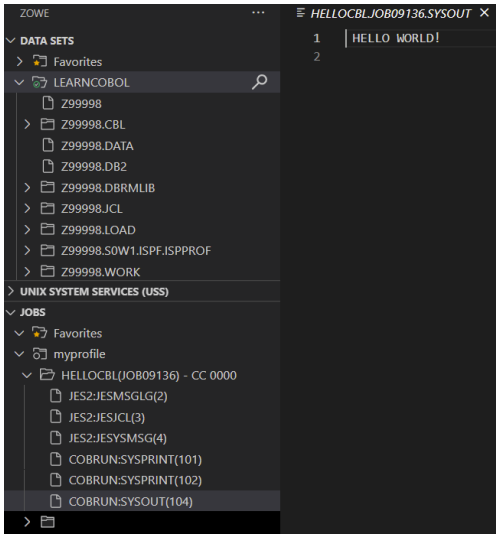


Figure 25. COBOL program execution

Copyright Contributors to the Open Mainframe Project's COBOL Programming Course

25. Do note that you will need to open the **LearnCOBOL** folder every time you connect to the system, repeating step 5 to 7. To enable your connection profile to be accessible anywhere on your machine, you will need to move your configuration files (i.e. `zowe.config.json` and `zowe.schema.json`) from the LearnCOBOL folder to the Zowe global location. By default this is `C:\Users\%USERNAME%\.zowe` for Windows or `~/.zowe` for Linux and macOS.

26. The following URL is another excellent document describing the above VS Code and Zowe Explorer details with examples: https://marketplace.visualstudio.com/items?itemName=Zowe.vscode-extension-for-zowe

2.7 LAB - ZOWE CLI & AUTOMATION

In this lab exercise, you will use the Zowe CLI to automate submitting the JCL to compile, link, and run the COBOL program and downloading the spool output. Refer to the section on the “Installation of Zowe CLI and Plug-ins” to install Zowe CLI if you have not already done so. Before developing the automation, we will first leverage the Zowe CLI interactively.

2.7.1 Zowe CLI - Interactive Usage

COBOL Programming Course #2 3.2.0|
250929-2138

In this section, we will use the Zowe CLI interactively to view data set members, submit jobs, and review spool output.

1. Within VS Code, open the integrated terminal (Terminal -> New Terminal). In the terminal, issue `zowe --version` to confirm the Zowe CLI is installed as depicted in the following figure. If it is not installed, please refer to the section on the “Installation of Zowe CLI and Plug-ins.” Also, notice that the default shell selected is `cmd`. I would recommend selecting the default shell as either `bash` or `cmd` for this lab.

   Figure 26. `zowe --version` command in VS Code Integrated Terminal

2. In order for Zowe CLI to interact with z/OSMF the CLI must know the connection details such as host, port, username, password, etc. While you could enter this information on each command, Zowe provides the ability to store this information in configuration files.

   If you have done the configuration in the first lab, you will have a folder containing your team configuration files. Make sure that your terminal is at that location and issue the following command.

   ```
   zowe config list --locations
   ```

The following figure demonstrates the outcome of the command.

Figure 27. List available team config connections

3. Confirm you can connect to z/OSMF by issuing the following command:

```
zowe zosmf check status
```

4. List data sets under your ID by issuing a command similar to (see sample output in the following figure):

```
zowe files list ds "Z99998.*"
```

You can also list all members in a partitioned data set by issuing a command similar to (see sample output in the following figure):

```
zowe files list am "Z99998.CBL"
```

Figure 28. zowe files list ds and am commands

5. Next, we will download our COBOL and JCL data set members to our local machine. From the terminal, issue commands similar to:

```
zowe files download am "Z99998.CBL" -e ".cbl"
zowe files download am "Z99998.JCL" -e ".jcl"
```

A completed example is shown in the following figure:

Figure 29. Download and view data set members using the CLI

6. Next, we will submit the job in member Z99998.JCL(HELLO). To submit the job, wait for it to complete, and view all spool content, issue:

```
zowe jobs submit ds "Z99998.JCL(HELLO)" --vasc
```

We could also perform this step in piecemeal to get the output from a specific spool file. See the next figure for an example of the upcoming commands. To submit the job and wait for it to enter OUTPUT status, issue:

```
zowe jobs submit ds "Z99998.JCL(HELLO)" --wfo
```

To list spool files associated with this job id, issue:

COBOL Programming Course #2 3.2.0|
25029-2138

```
zowe jobs list sfbj JOB04064
```

where JOB04064 was returned from the previous command.

To view a specific spool file (COBRUN:SYSOUT), issue:

```
zowe jobs view sfbi JOB04064 105
```

where JOB04064 and 105 are obtained from the previous commands.

Figure 30. Submit a job, wait for it to complete, then list spool files for the job, and view a specific spool file

If desired, you can also easily submit a job, wait for it to complete, and download the spool content using the following command (see the following figure for the completed state):

```
zowe jobs submit ds "Z99998.JCL(HELLO)" -d .
```

Figure 31. Submit a job, wait for it to complete, download and view spool files

The Zowe CLI was built with scripting in mind. For example, you can use the --rfj flag to receive output in JSON format for easy parsing. See the next figure for an example.

Figure 32. The --rfj flag allows for easy programmatic usage

COBOL Programming Course #2 3.2.0|
250929-2138

2.7.2 Zowe CLI - Programmatic Usage

In this section, we will leverage the Zowe CLI programmatically to automate submitting the JCL to compile, link, and run the COBOL program and downloading the spool output. Once you have the content locally you could use any number of distributed scripting and testing tools to eliminate the need to manually review the spool content itself. Historically, in Mainframe, we use REXX Exec, CLIST, etc. for automation, but today we are going to use CLI and distributed tooling.

1. Since we already have Node and npm installed, let's just create a node project for our automation. To initialize a project, issue npm init in your project's folder and follow the prompts. You can accept the defaults by just pressing enter. Only the description and author fields should be changed. See the following figure. Do note that to use the team configuration files as mentioned on previous section, this project must be a subdirectory of the **LearnCOBOL** folder.

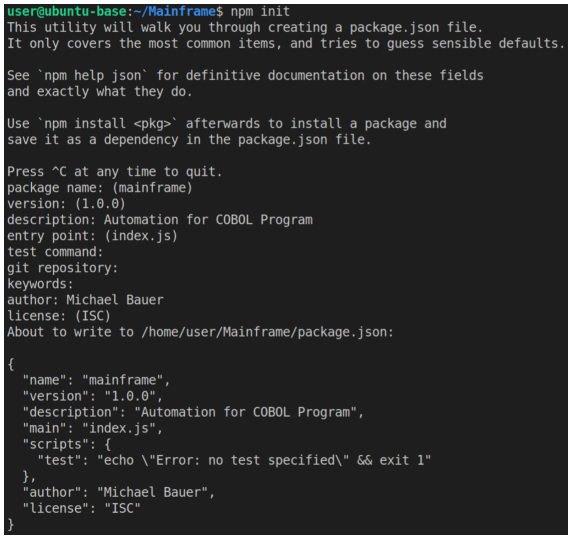


Figure 33. Use of npm init to create package.json for the project

2. Now that we have our package.json simply replace the test script with a clg script that runs the following zowe command (replace Z99998 with your high-level qualifier):

```
zowe jobs submit ds 'Z99998.JCL(HELLO)' -d .
```

You can name the script whatever you want. I only suggested clg because the CLG in the IGYWCLG proc (which is what the JCL leverages) stands for compile, link, go. Now, simply issue npm run clg in your terminal to leverage the automation to compile, link, and run the COBOL program and download the output for review. An example of the completed package.json and command execution are shown in the following figure.

COBOL Programming Course #2 3.2.0| 250929-2138

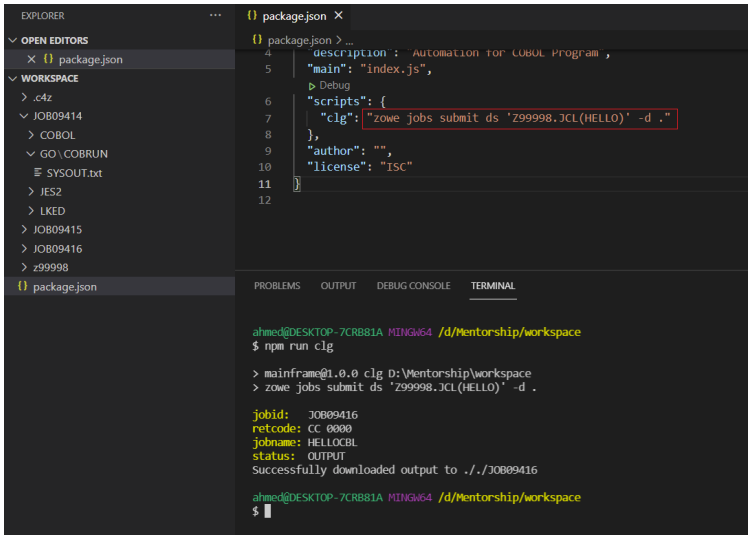


Figure 34. Final package.json and npm run clg execution

3. If you prefer a graphical trigger, you can leverage VS Code as shown in the following figure. Essentially, the CLI enables you to quickly build your own buttons for your custom z/OS tasks. You could also invoke a script rather than a single command to accommodate more complex scenarios.

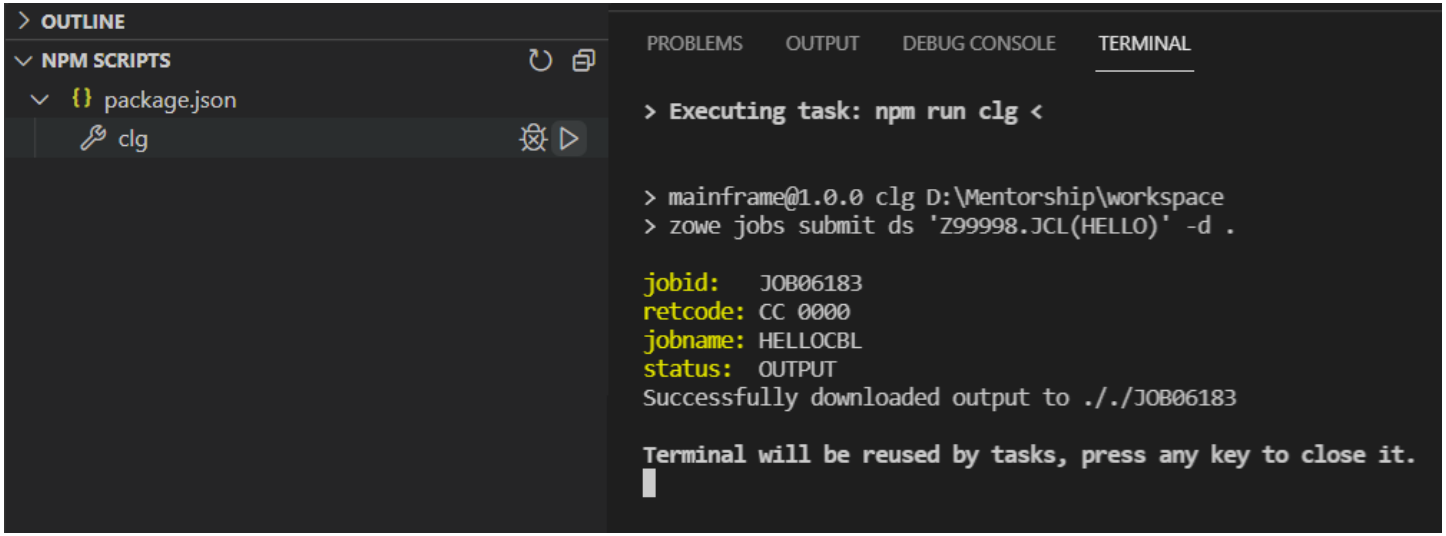


Figure 35. clg task triggered via button

Copyright

COBOL Programming Course is licensed under Creative Commons Attribution 4.0 International.

To view a copy of this license, visit https://creativecommons.org/licenses/by/4.0.

Copyright Contributors to the Open Mainframe Project's COBOL Programming Course

COBOL Programming Course #2 3.2.0|
250929-2138