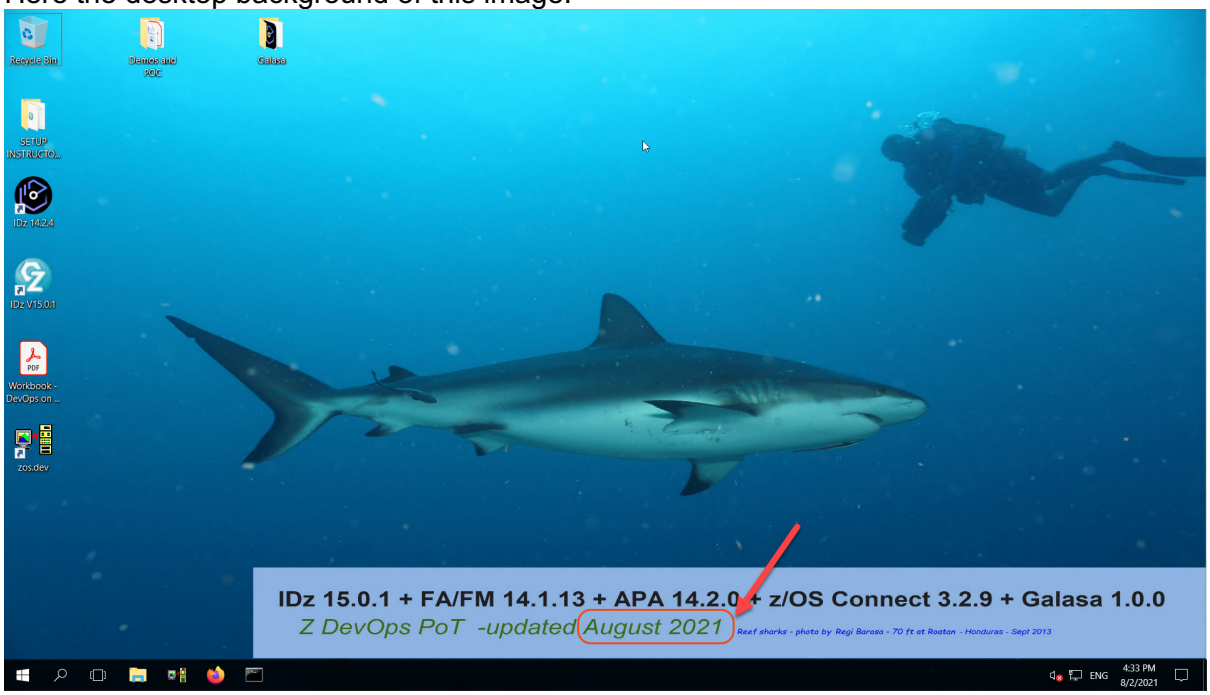
DevOps on Z
Proof of Technology (PoT)

Overview

- Integrated application development and problem analysis (ADFz)
- Managing your source code using modern tools (including Git)
- Building automation (DBB) for COBOL or PL/1 without a specific source code manager or pipeline automation tool (including Groovy and Jenkins)
- Using a unit testing framework (zUnit) for COBOL or PLI as part of the development environment
- Using IBM Z Virtual Test Platform (VTP) for application Integration Testing of a COBOL/CICS/DB2 application
- Using GitLab CI along with DBB, ZUnit and UrbanCode Deploy (UCD) on z/OS.
- Integration testing for z/OS powered hybrid cloud applications using Galasa
- Application deployments automation to many environments (UCD)
- Expose Mainframe legacy applications via RESTful APIs (z/OS Connect)
- Understand how Mainframe applications use their resources to improve performance (APA)

This material was built using a Windows on cloud that was updated on **August 2021**
Here the desktop background of this image:



The following symbols appear in this document at places where additional guidance is available.

Icon	Purpose	Explanation
	Important!	This symbol calls attention to a particular step or command. For example, it might alert you to type a command carefully because it is case sensitive.
	Information	This symbol indicates information that might not be necessary to complete a step, but is helpful or good to know.
	Trouble-shooting	This symbol indicates that you can fix a specific problem by completing the associated troubleshooting information.

LAB 1 - Working with mainframe using COBOL and DB2
(90 - 120 minutes)

Updated September 24, 2021 (created by Regi,, Reviewed by Wilbert

This lab will take you through the steps of using the Application Delivery Foundation for z Systems (ADFz) to work with a z/OS system. It will familiarize you with some of the capabilities of this product using a DB2 COBOL batch program that is ABENDing.

On this lab you will connect to a remote z/OS system, submit and execute a program (which ABENDs), identify the program abend, set up a MVS project, edit, compile, and debug a COBOL application.
The process would be similar for a PL/I program.

Overview of development tasks

To complete this tutorial, you will perform the following tasks:

1. **Connect to a z/OS System.**
   → You will connect to the z/OS system using a provided z/OS logon userid.
2. **Execute the DB2/COBOL batch program and verify the ABEND.**
   → You will submit a COBOL/DB2 batch to execute and verify the bug.
3. **Use Fault Analyzer to identify the cause of the ABEND**
   → You will use Fault Analyzer to identify what is causing the ABEND.
4. **Use the IBM Debug for a temporary fix**
   → You will modify the field content to bypass the bug
5. **Modify the COBOL code to fix the bug.**
   → You will be able to fix the bug changing the COBOL code.
6. **Use Code Coverage**
   → You can now verify program areas covered by the test case with Code Coverage
7. **(Optional) Execute SQL statement when editing the program.**
   → While editing the COBOL/DB2 program you will be able to execute SQL statements and verify the query results.
8. **(Optional) Using File Manager**
   → An example of using File Manager against a z/OS data set.
   File Manager provides formatted editors and viewers. It also provides a full complement of on-line and batch utilities to copy, extract, and load data, to create files and databases, compare and print, and many other utility functions.

Section 1 – Connect to a z/OS System

You will connect to the z/OS. This section is like LOGON to a TSO. You will use **empot01** as userid and password.

___ 1.1 Working with the IDz workspace

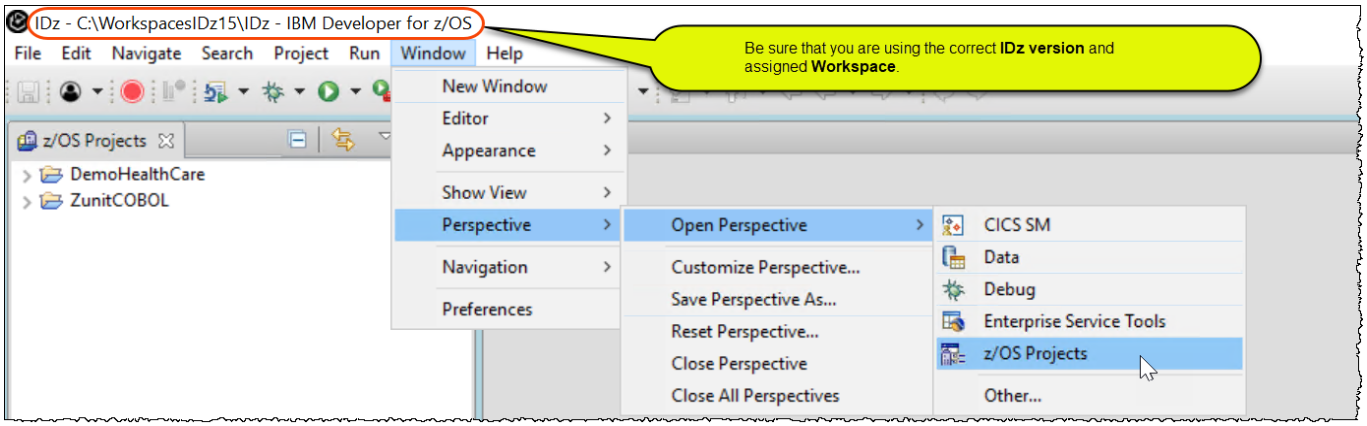
1.1.1 Start *IBM Developer for z Systems version 15*

⏭ Using the windows desktop double click on **IDz V15** icon.

⏭ Verify that the message indicates that it is *Version **15.0.1***

IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon.

1.1.2 ⏭ Open the **z/OS Projects** perspective by selecting **Window > Perspective > Open Perspective > z/OS Projects**



z/OS Projects perspective

Use the z/OS Projects perspective to define the connection, connect to, and work with remote systems, and to create, edit, and build projects, subprojects, and files on remote UNIX, Linux for z Systems, and z/OS systems.

The z/OS Projects perspective contains the many views like Remote Systems view, z/OS Projects view, Properties view, Outline view, Remote Error List view, z/OS File System Mapping view, Remote System Details view, Team view, Property Group Manager view and Snippets view

You can close and open views to customize the perspective.

To open one of the views that were closed:

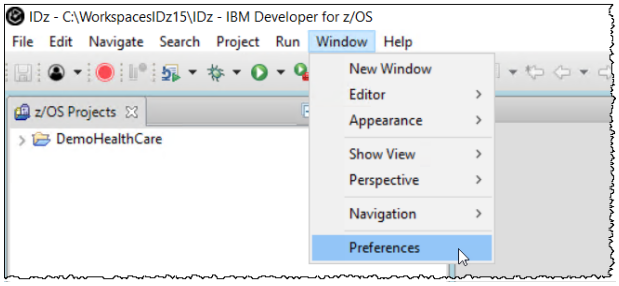
1. In the workbench, select **Window** > **Show View**.

A menu that lists the views associated with the z/OS Projects perspective is displayed.

2. Click the name of the view you want to open.

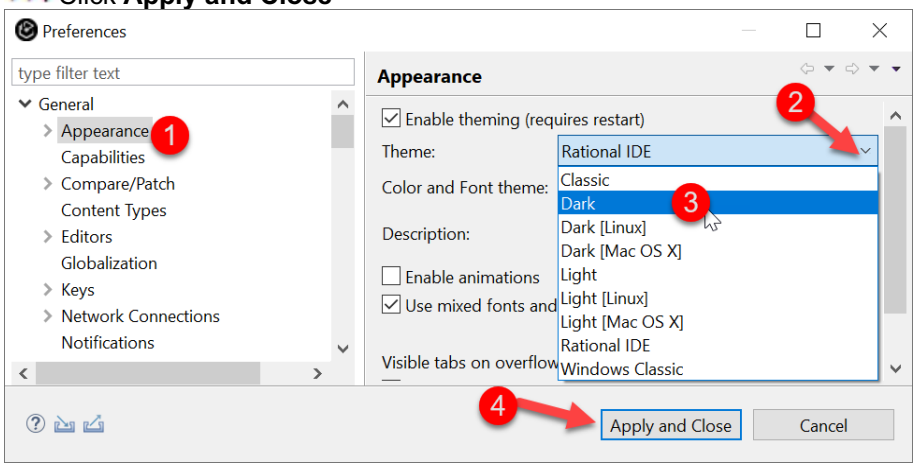
1.1.3 Starting on version 15 the developer can change the workspace appearance. If you want to try that

⏭ Click **Windows > Preferences**



1.1.4 ⏭ Under **General** select **Appearance** and on *Theme* select **Dark**.

⏭ Click **Apply and Close**

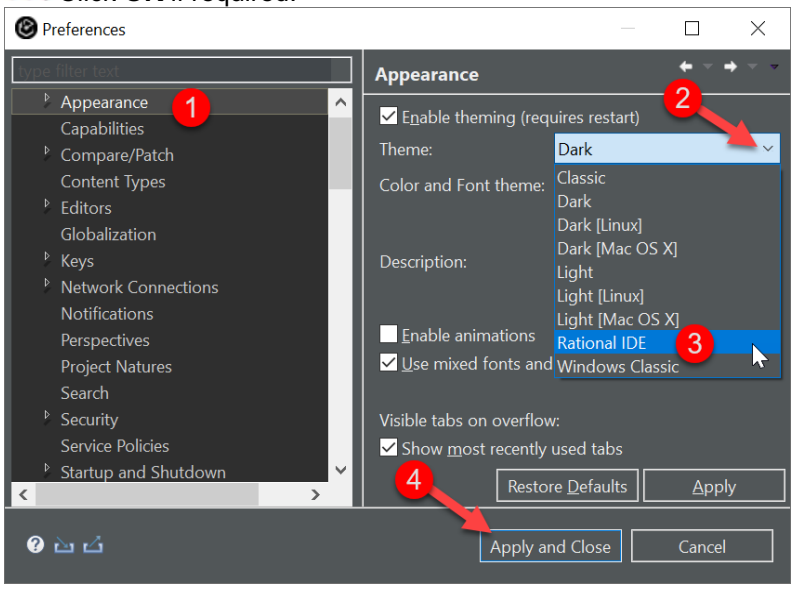


1.1.5 Since all picture here were done using “Rational IDE” theme we recommend to return to the default.

⏭ Again, click **Windows > Preferences**

⏭ Under **General** select **Appearance** and on *Theme* select **Rational IDE** and click **Apply and Close**

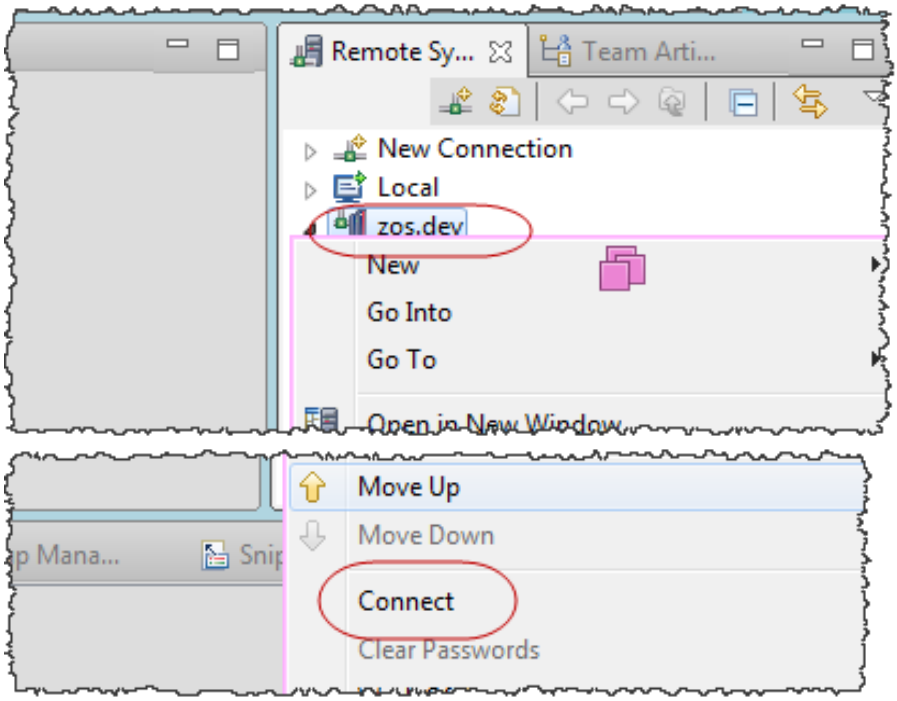
⏭ Click **OK** if required.



___ 1.2 Connecting to the z/OS Remote system

1.2.1 To Connect to the z/OS system:

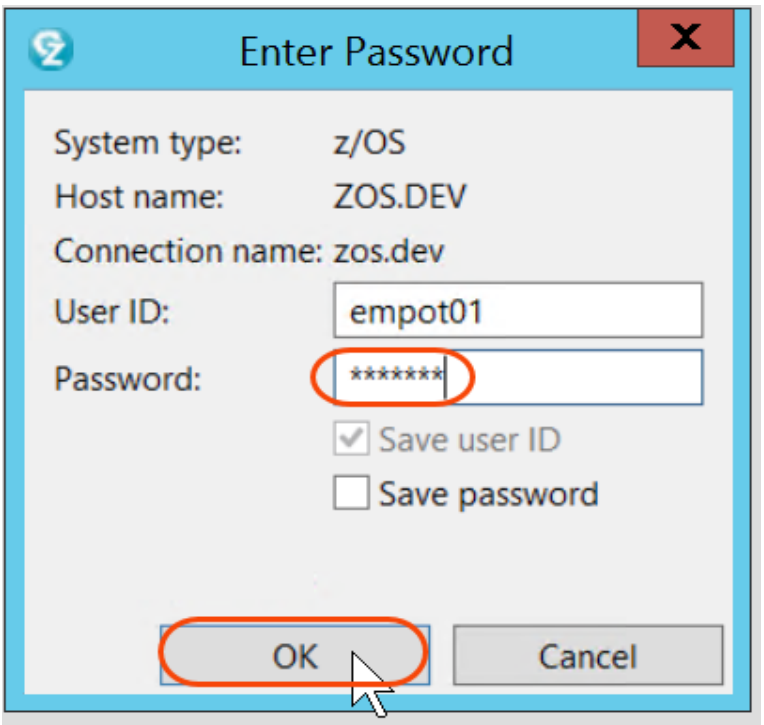
⏭ Using the *Remote Systems view* on the top and right side of the screen:
Right-click on **zos.dev** and select **Connect**
Notice: Since you are using a cloud instance the response to the right click may be delayed first time.



1.2.2 You will be are prompted for your z/OS userid and password.

⏭ Type **empot01** as userid (might be already there) and **empot01** as password.
The userid and password can be any case; don't worry about having it in UPPER case.

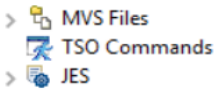
1.2.3 ⏭ Click **OK** to connect to z/OS.

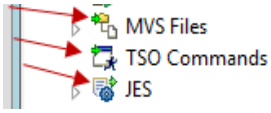


Be Patient! The connection could take a while depending on the network condition.

Wait until connection is complete (look at the bottom and left until the green bar disappears)

Notice that some folder icons changed after connected. A small **green arrow** is added.

BEFORE… 

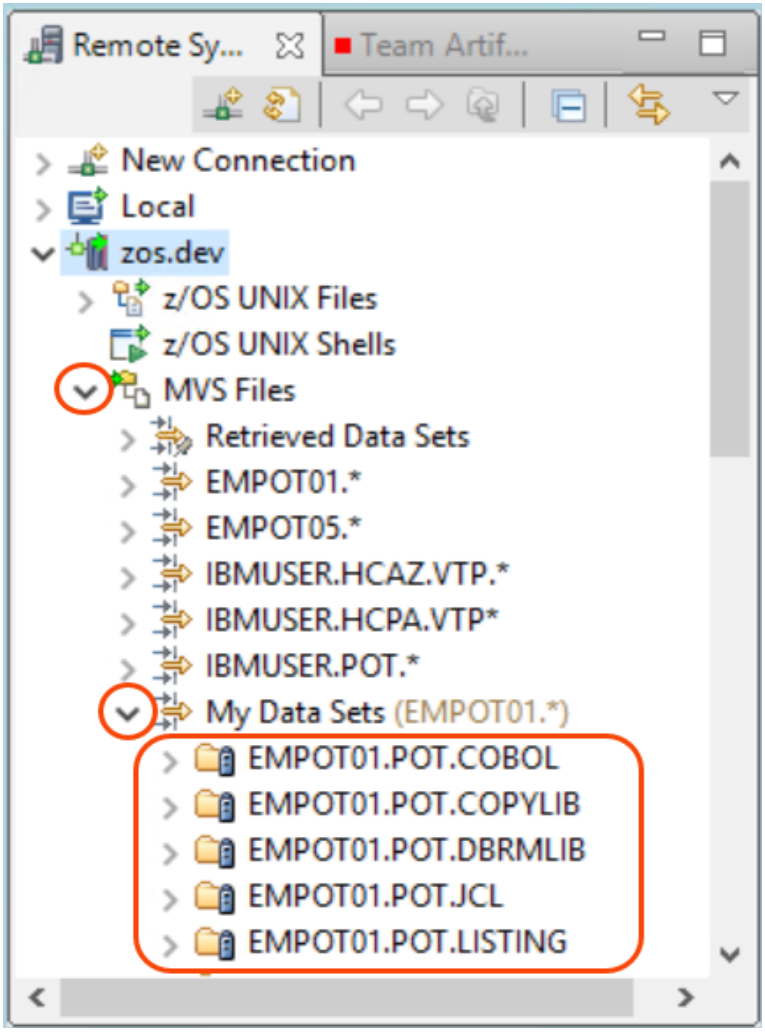
AFTER: 

In case of connection errors…

If you have errors during the connection it is because the z/OS system is not available.
You also may have network issues.

An easy way to find if z/OS is up is opening a 3270 emulation clicking on the icon that is on the base of your windows client. (zos.dev is not "pingable" .
If you do not receive a reply, Contact the instructor. Your connection is broken.

1.2.5 Expand MVS Files and My Data Sets to see MVS data sets that starts with EMPOT01.*



Remote Systems view

The Remote Systems view shows all existing connections to remote systems.
Connections are persisted, containing the information needed to access a remote host.
The view contains a prompt to create new connections, and pop-up menu actions to rename, copy, delete, and reorder existing connections.
Connections contain attributes, or data, that are saved between sessions of the workbench. These attributes are the connection name, the remote system's host name and system type, an optional description, and a user ID that is used by default by each subordinate subsystem, at connection time.

Section 2 Execute the DB2/COBOL batch program and verify the ABEND.

To make your job easier, you will group together all the assets that you will work with. This is a new development concept for TSO/ISPF users, since TSO/ISPF does not provide such capability. To accomplish this task, you will create a **z/OS project** and select which assets will be part of this project.

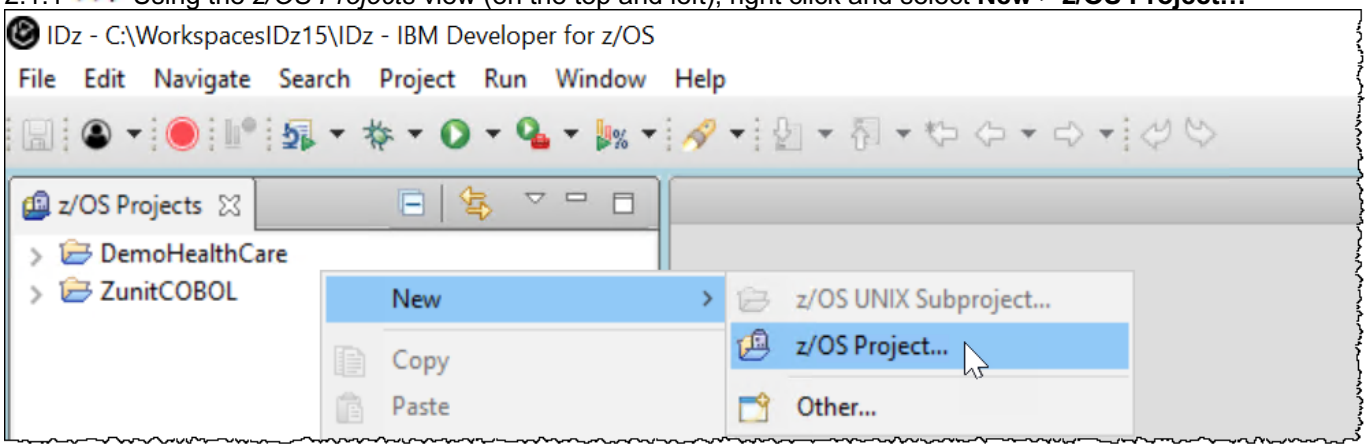
What is a z/OS project?
After you define a z/OS-connection and connect to that system, you can define a z/OS project under that system. You can define the z/OS project, however, only when you are connected to the system. Application Delivery Foundation for z Systems assigns a set of default properties from the set of system properties. However, changes that you make to system properties do not affect the definition of an existing project. If you change your system properties to reference a new compiler release, for example, the reference affects only those projects that are defined subsequent to the change. This isolation of system data from existing projects is beneficial because it lets you develop your code without disruption. You can introduce changes to the project definition at a time of your choosing.

Creating a new MVS subproject
MVS subprojects contain the development resources that reside on an MVS system. You can create multiple subprojects in a z/OS project.
Before you create an MVS subproject, you need to have completed the following tasks:
→ Connecting to a remote system
→ Creating a z/OS project

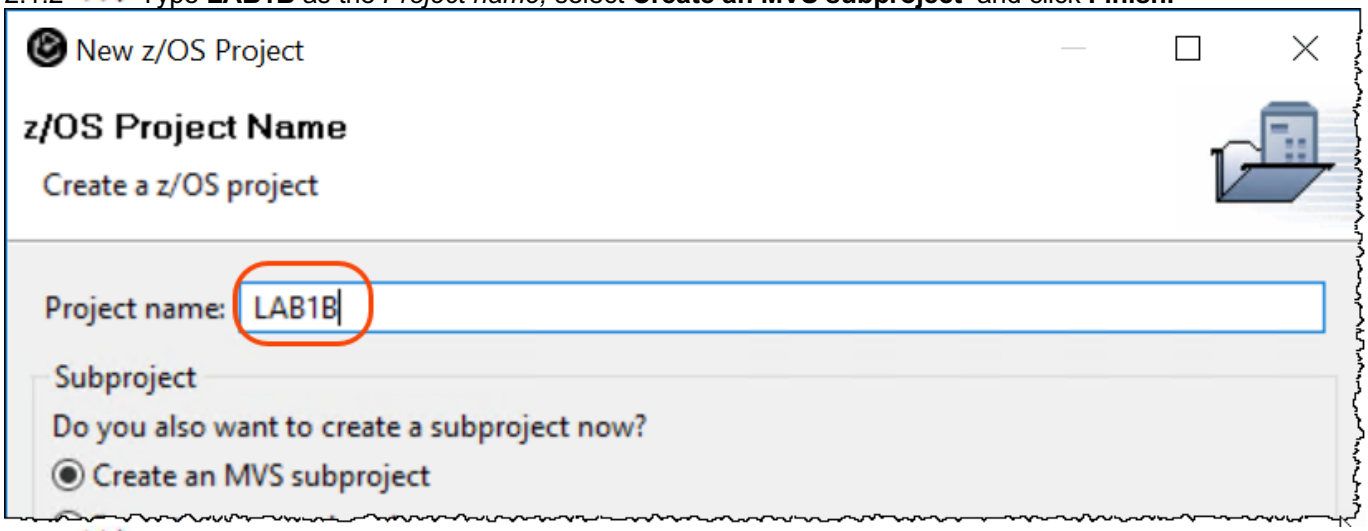
___ 2.1 Submit a COBOL/DB2 batch to execute

The advantage of creating a *z/OS Project* is that we just focus on those datasets and members that are being constructed or updated, instead of having all the mainframe datasets or members. At any time if you need to see a dataset not added to the project, just go to the z/OS Systems view and add it. In addition, at any time, you can remove from your project the datasets that you don't need anymore.

2.1.1 Using the *z/OS Projects* view (on the top and left), right click and select **New** > **z/OS Project…**

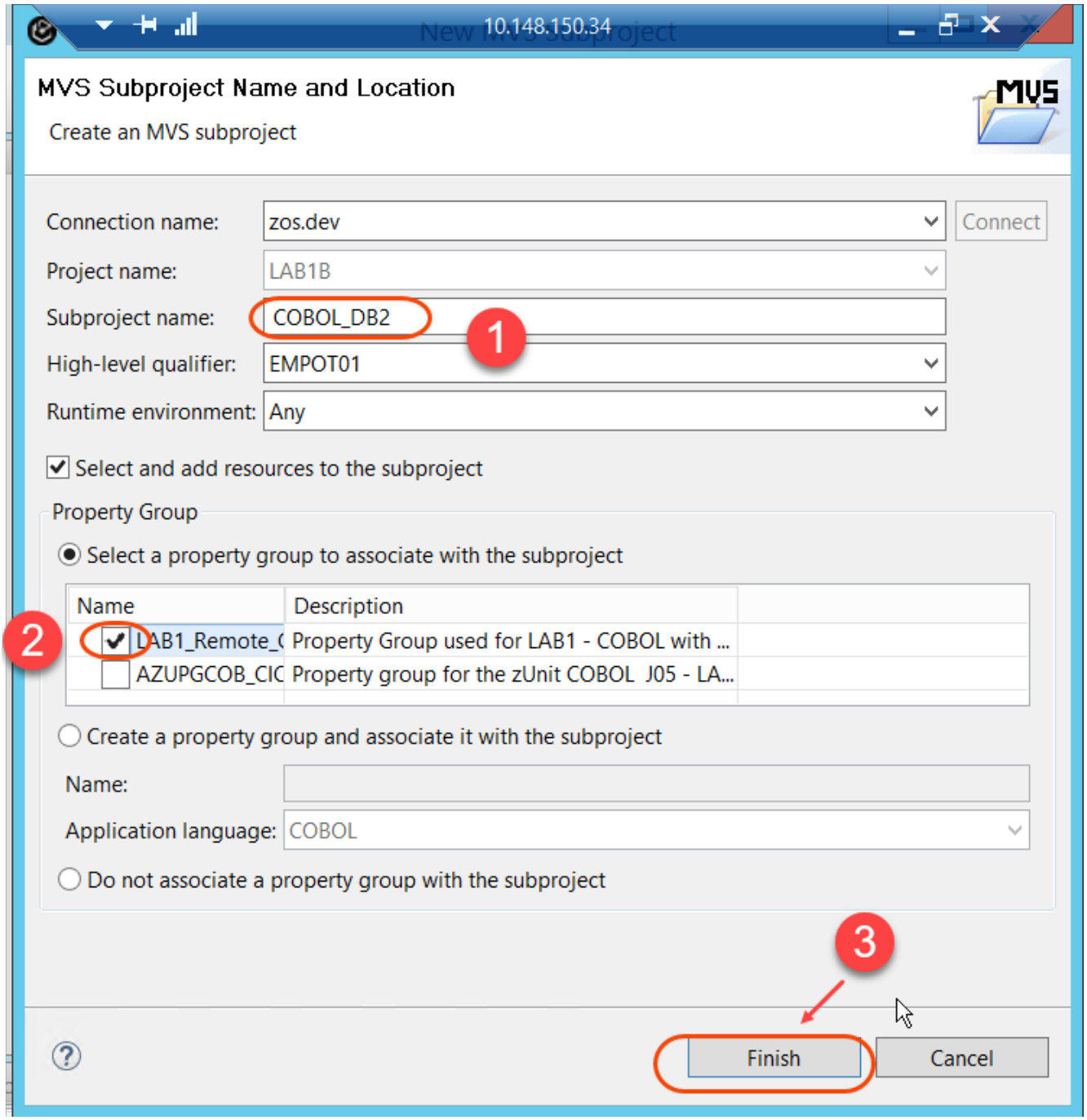


2.1.2 Type **LAB1B** as the *Project name,* select **Create an MVS subproject** and click **Finish.**

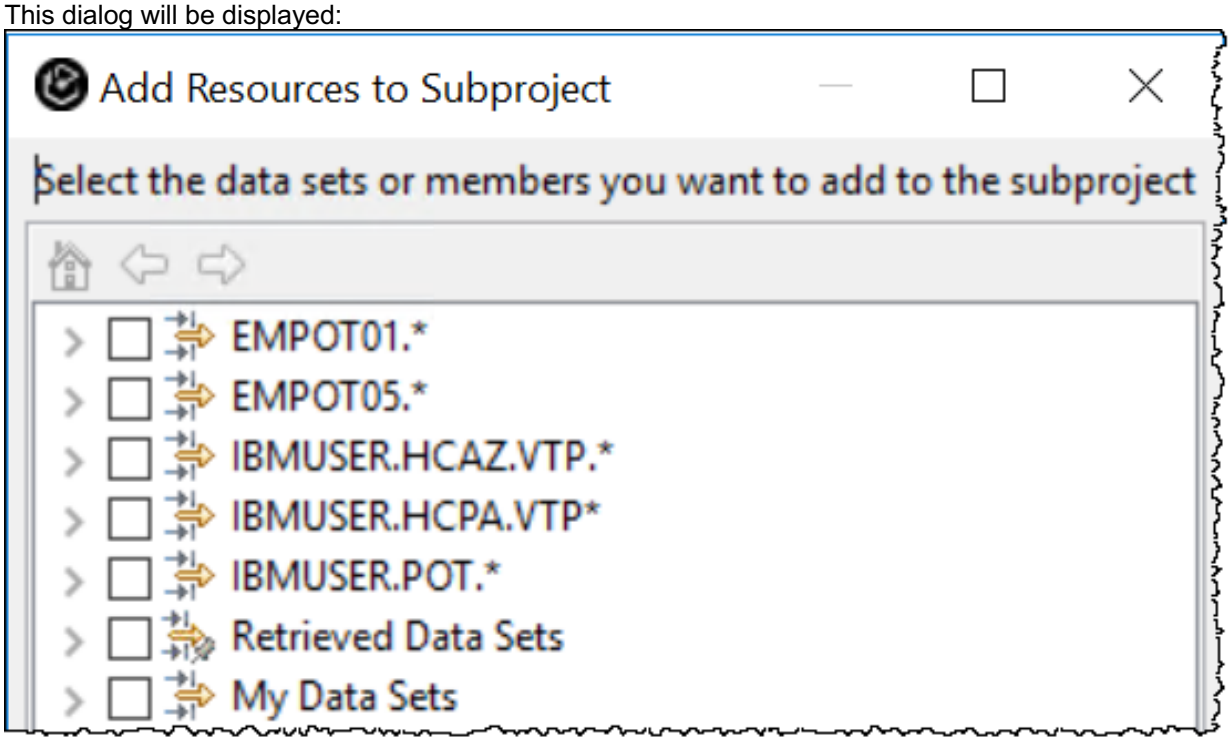


2.1.3 Type **COBOL_DB2** as *Subproject Name,* select **LAB1_Remote_Cobol..** as the property group and click **Finish .**

TIP: If using *Spanish or Brazilian* keyboard you must activate the correct language in the bottom, right corner, otherwise you will not be able to generate the "_".


10.148.150.34
MVS Subproject Name and Location
Create an MVS subproject
Connection name:
zos.dev
Connect
Project name:
LAB1B
Subproject name:
COBOL_DB2
1
High-level qualifier:
EMPOT01
Runtime environment:
Any
Select and add resources to the subproject
Property Group
Select a property group to associate with the subproject
Name
Description
2
LAB1_Remote_C
Property Group used for LAB1 - COBOL with ...
AZUPGCOB_CIC
Property group for the zUnit COBOL J05 - LA...
Create a property group and associate it with the subproject
Name:
Application language:
COBOL
Do not associate a property group with the subproject
3
Finish
Cancel

This dialog will be displayed:



Can't you find the property group above?

It is because you are using a wrong workspace. Close IDz, go back to the step 1.1.1 and be sure you start IDz from the icon that is on the windows desktop.

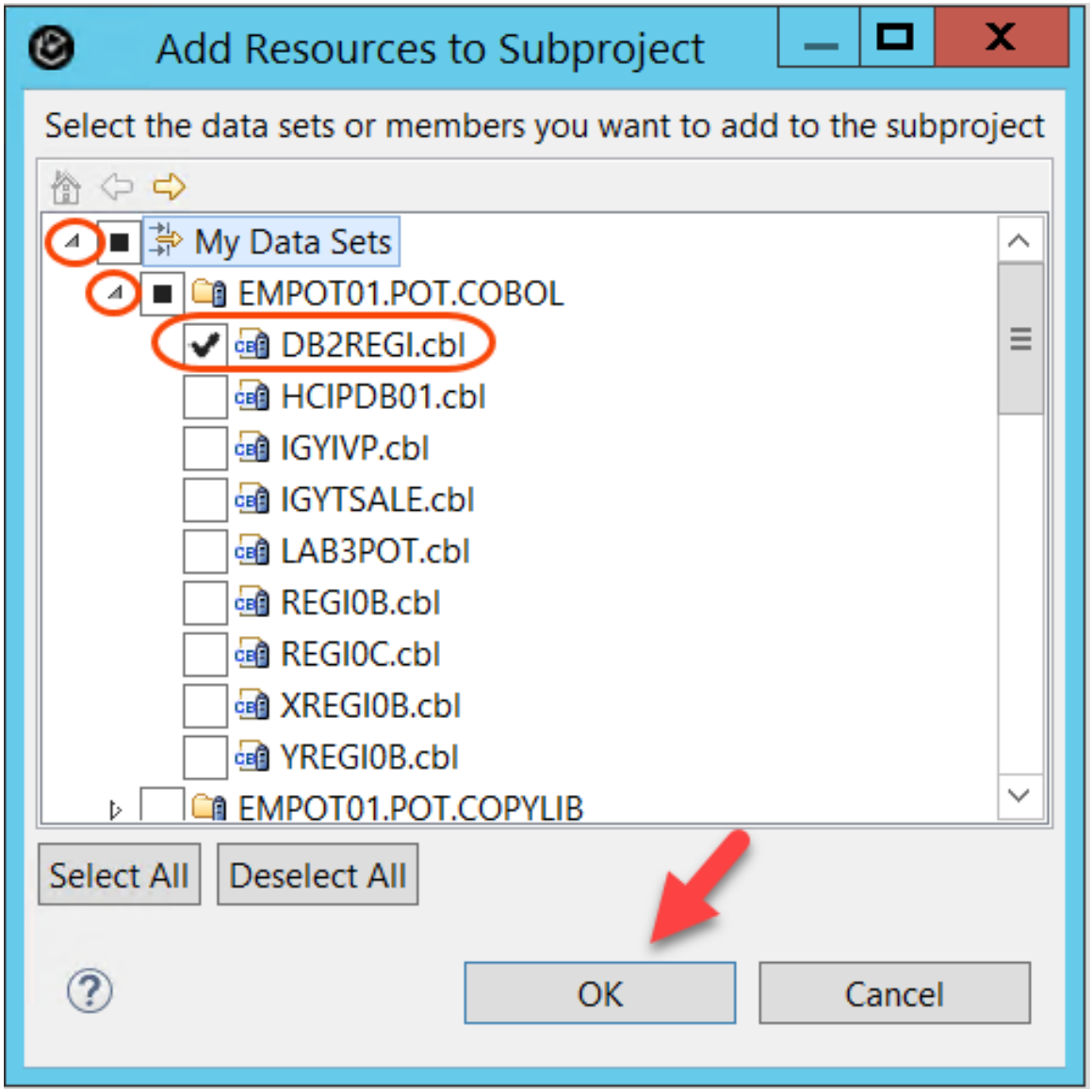
– PROPERTY GROUP

Property groups provide a way to manage resource properties, share them easily across systems, projects, resources, and users, and maintain consistency in your development and build environment.
You can, for example, define a property group that points to copybooks. This is something like `//STEPLIB`; otherwise, some of the variables used in the program are not resolved.

___ 2.2 Add z/OS resources to the MVS subproject

To make the data sets available to your remote project named *LAB1B,* you will need to add them. For this lab we just want to add one member..

2.2.1 ⏭ On the dialog below, expand **My Data Sets, EMPOT01.POT.COBOL ,** select **DB2REGI.cbl,** and click **OK**

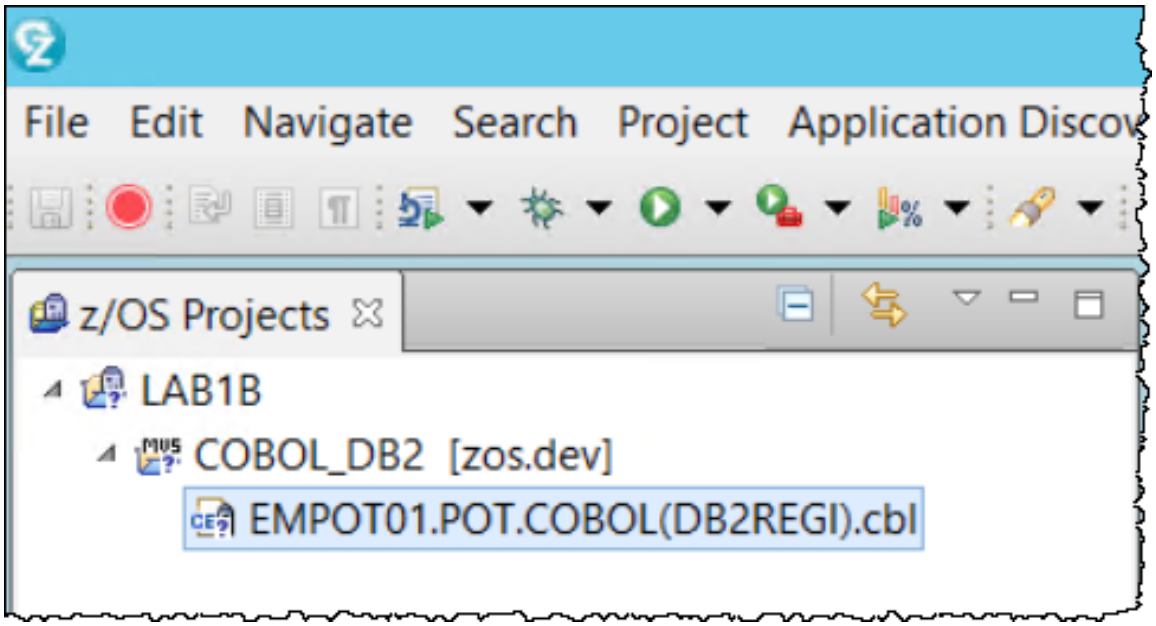

Add Resources to Subproject
Select the data sets or members you want to add to the subproject
My Data Sets
EMPOT01.POT.COBOL
DB2REGI.cbl
HCIPDB01.cbl
IGYIVP.cbl
IGYTSALE.cbl
LAB3POT.cbl
REGI0B.cbl
REGI0C.cbl
XREGI0B.cbl
YREGI0B.cbl
EMPOT01.POT.COPYLIB
Select All
Deselect All
OK
Cancel

2.2.2 You should see a z/OS Project named **LAB1B** in your *z/OS Projects view*.

⏭ Click ***z/OS Projects view*** (on left) and under **COBOL_DB2,** you will now see **DB2REG**I member added to your project.

Notice that creating a Remote project is a good practice to isolate the code you are working on, but you could work on your code without creating this project.

Our lab involves a small update so you could work directly on the *Remote System* perspective without having to create a Remote project..



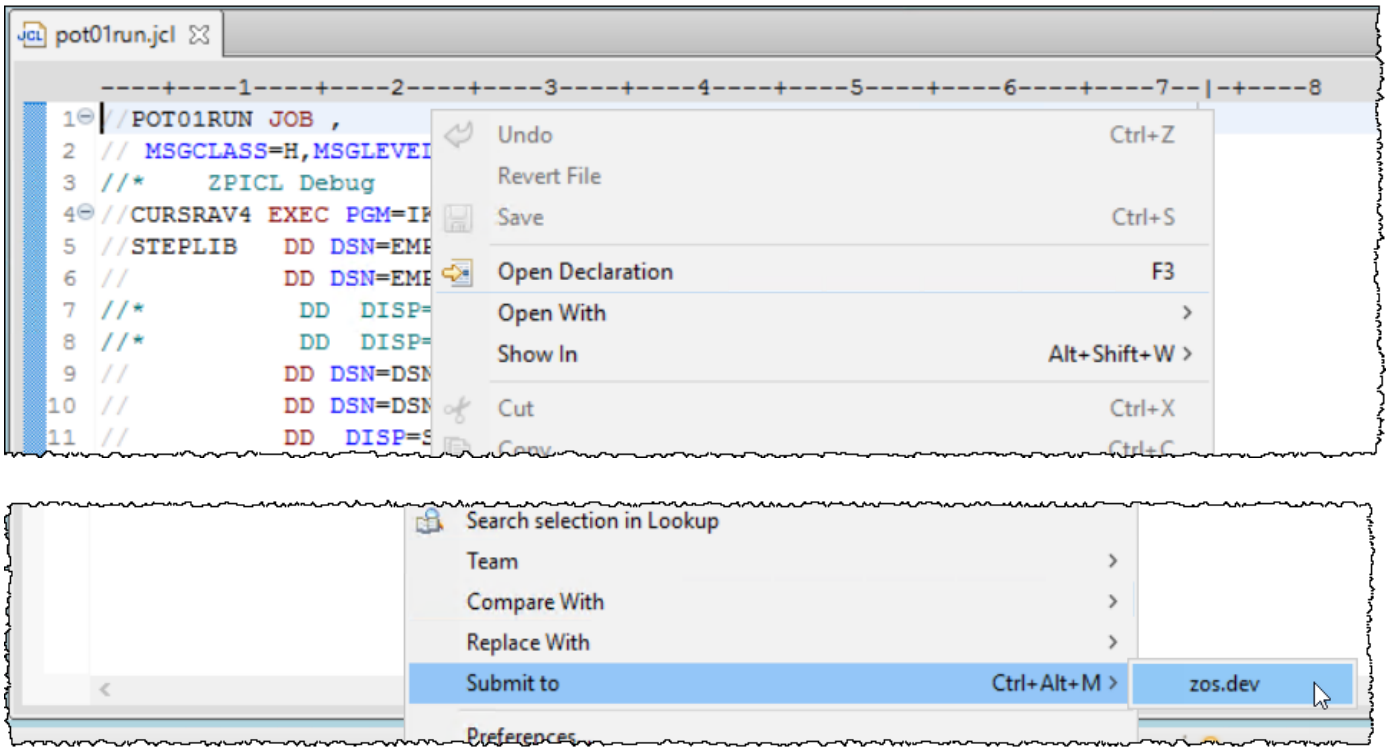
___ 2.3 Submit a JCL to execute the COBOL program

The JCL to execute the program is available on the local windows folder: "**`C:\ADF_POT\empot01`**".
You will use this JCL to execute a batch COBOL/DB2 on z/OS.

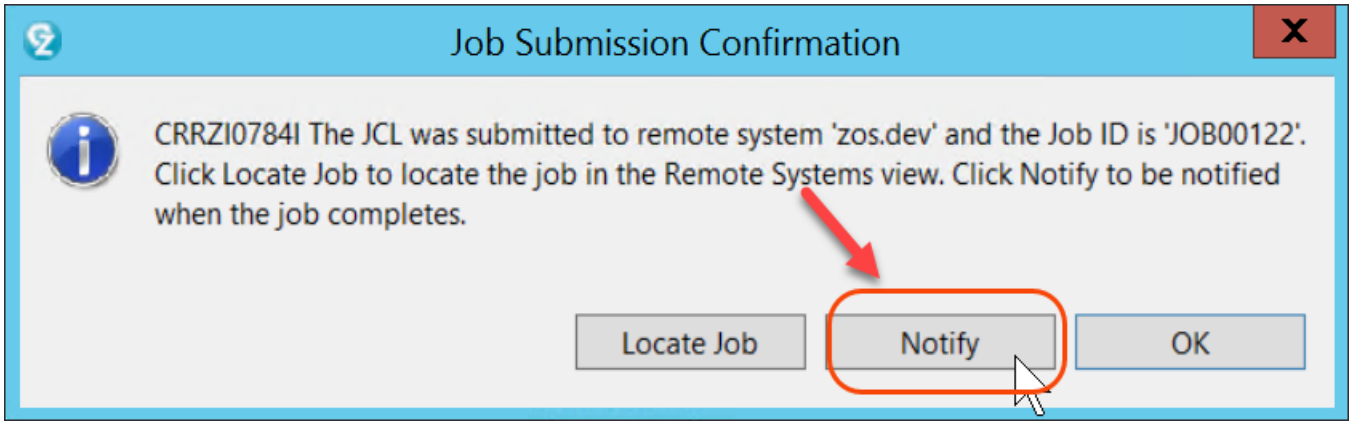
2.3.1 ⏭ Using *Remote System View* (on top and right), **scroll up** to locate the file **`pot01run.jcl`** under **`Local/Local Files/ADF_POT/empot01`** and **double click** to edit.


pot01run.jcl
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
//POT01RUN JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
//*   ZPICL Debug
//CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB   DD DSN=EMPOT01.POT.LOAD,DISP=SHR
//   DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR
//*   DD DISP=SHR,DSN=IBMUSER.VER1.SEQAMOD
//*   DD DISP=SHR,DSN=IBMUSER.VER1.SEQAAUTH
//   DD DSN=DSNB10.SDSNLOAD,DISP=SHR
//   DD DSN=DSNB10.DBBG.RUNLIB.LOAD,DISP=SHR
//   DD DISP=SHR,DSN=FEK910.SFEKAUTH
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN   DD *
  TSOLIB ACTIVATE DA('DSNB10.SDSNLOAD')
  DSN SYSTEM(DBBG)
  RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -
  LIB('EMPOT.ZPICL.LOAD')
END
Be sure that yoy are editing pot01run.jcl
(NOT pot01db.jcl)
Remote Systems
Team Artifacts
New Connection
Local
Local Files
My Home
Drives
ADF_POT
empot01
alloc01.jcl
pot01db.jcl
pot01run.jcl
empot02
empot05
ibmuser
LAB1
LAB2A
LAB2B
LAB2C
LAB2D
LAB6

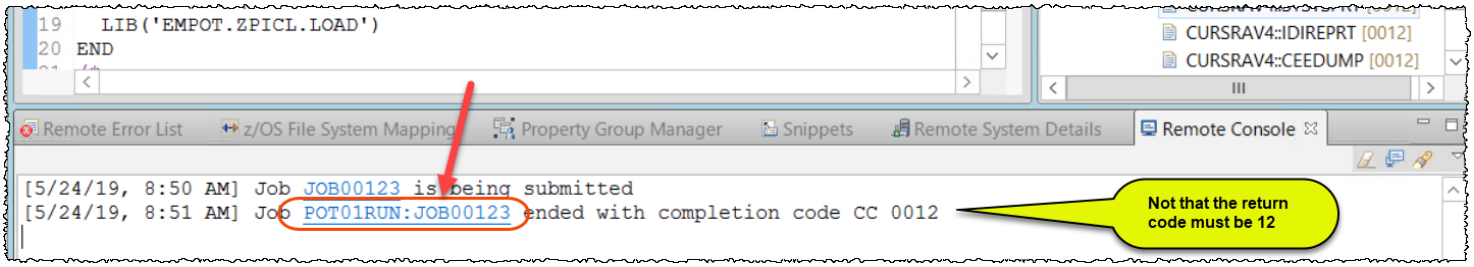
2.3.2 **Right click** on the JCL being edited and select **Submit to → zos.dev**



2.3.3 When the dialog pop up appears, click **Notify.** This allows you to be notified when the execution is ended.

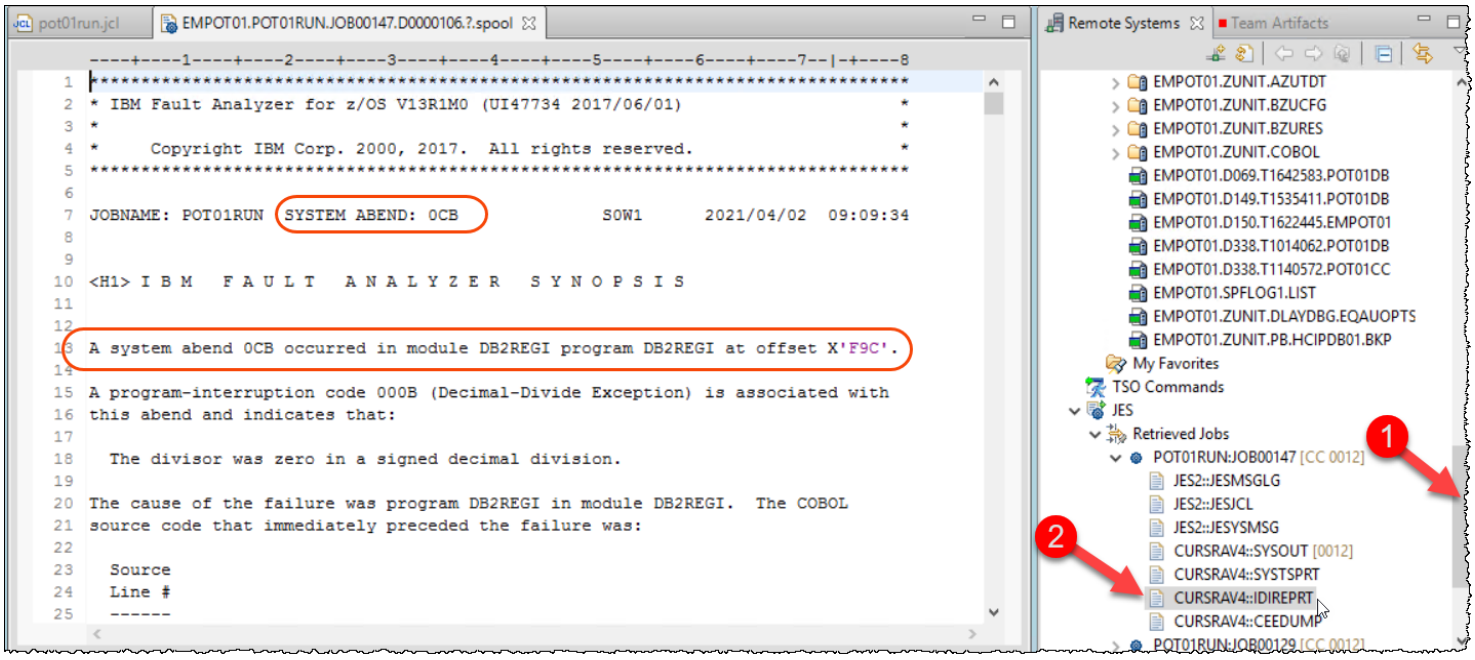


2.3.4 Under *Remote Console*, you will be notified when execution is completed.
Once the execution ends, **click** on the link **POT01RUN:JOB00xxx**
(where 00x**xx** is a number) to point to the Job output. This number vary depending on z/OS.

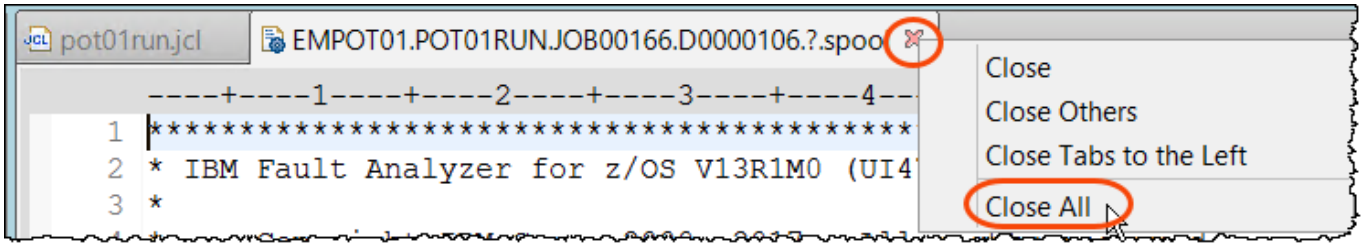

19 LIB('EMPOT.ZPICL.LOAD')
20 END
CURSRAV4::IDIREPRT [0012]
CURSRAV4::CEEDUMP [0012]
Remote Error List
z/OS File System Mapping
Property Group Manager
Snippets
Remote System Details
Remote Console
[5/24/19, 8:50 AM] Job JOB00123 is being submitted
[5/24/19, 8:51 AM] Job POT01RUN:JOB00123 ended with completion code CC 0012
Not that the return code must be 12

2.3.5 Under *Remote Systems* view **scroll down**, expand **JES > Retrieved Jobs > POT01RUN:JOB00xxx** and **double click CURSRAV4::IDIREPRT** step and you will see the *Fault Analyzer* report showing an ***OCB*** ABEND. Your mission is to fix that bug.

Tip: If there is no jobs under "*Retrieved Jobs"*, is because you did not click on link as stated at 2.3.4.
You can also see this output once you right click on "*My Jobs*" under *JES* and select ***Refresh***.



2.3.6 Close all the opened editors using **Ctrl + Shift + F4,**
Or right click on the . and choose **Close all.**



Section 3. Use Fault Analyzer to identify the cause of the ABEND

You will now take advantage of **IBM Fault Analyzer** (that is part of Application Delivery Foundation- ADF) to verify the cause of the ABEND.

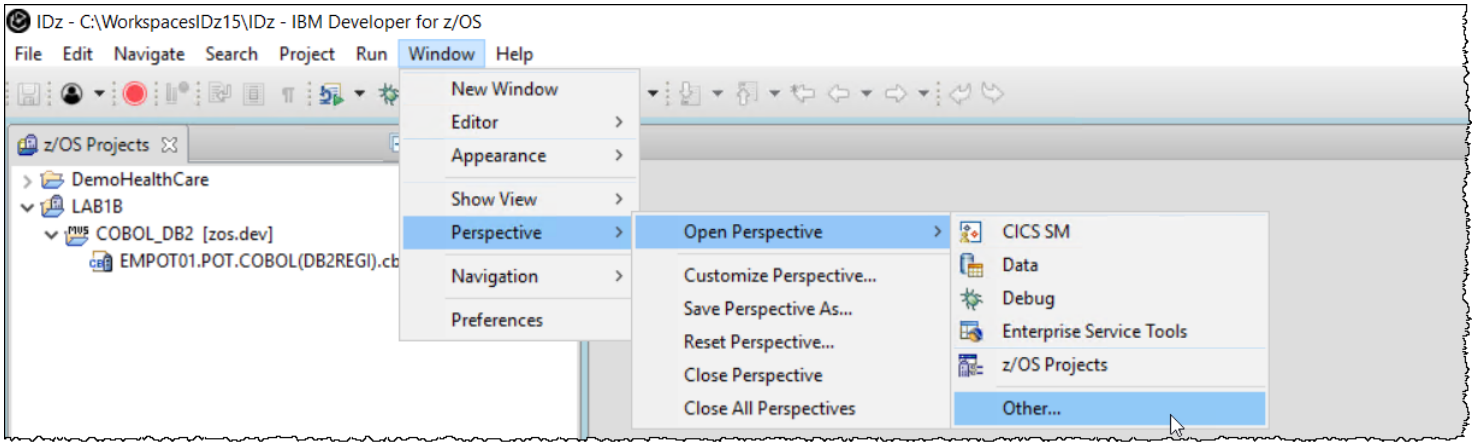
What is IBM Fault Analyzer for z/OS?

IBM Fault Analyzer for z/OS helps developers analyze and fix system and application failures for CICS, WebSphere MQ, IMS and DB2 environments. When an application ends abnormally, Fault Analyzer is automatically engaged, gathering real-time information about the event and its environment at the time of failure.

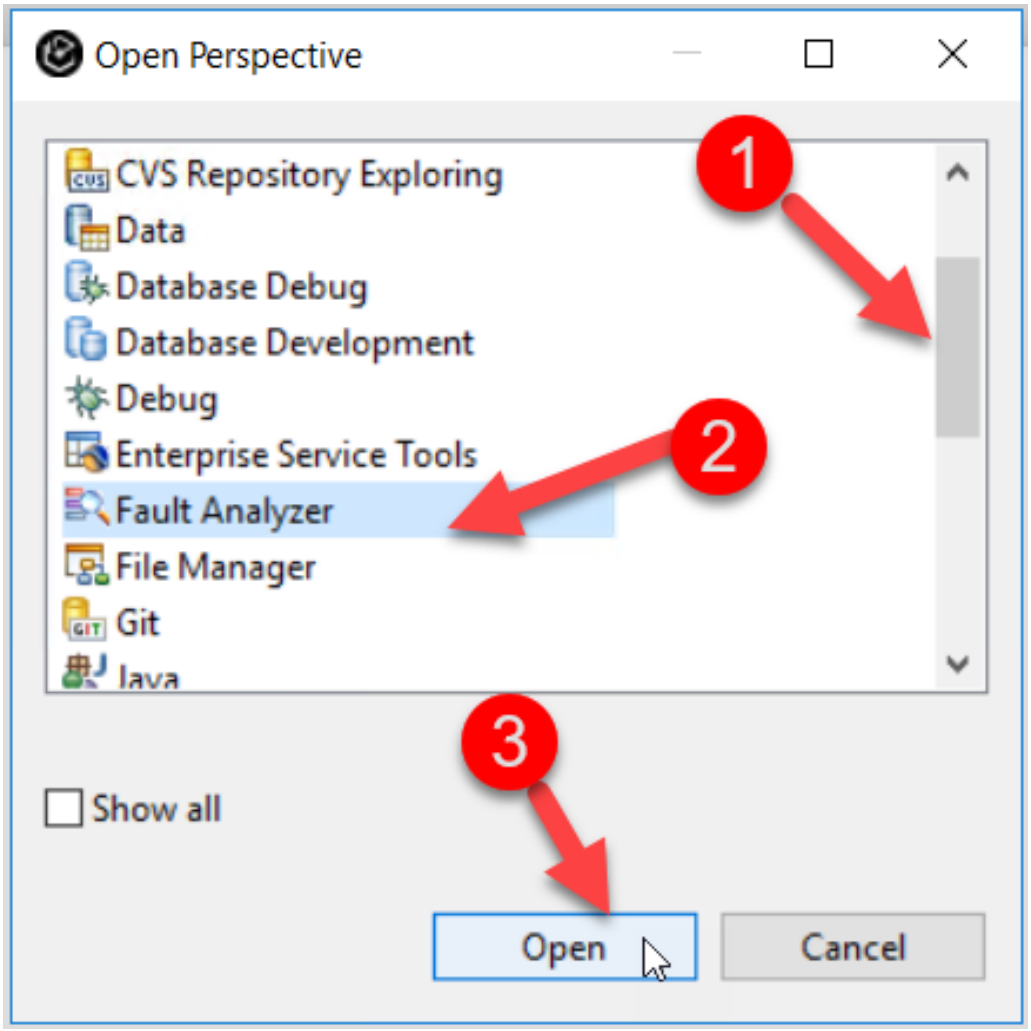
This helps the development team to identify the cause, analyze what went wrong and resolve the problem in a more timely and efficient manner to avoid costly interruptions that could jeopardize application schedules and outcomes.

___ 3.1 Using Fault Analyzer perspective

3.1.1 ⏭ Open the *Fault Analyzer* perspective using **Window > Perspective > Open Perspective > Other…**

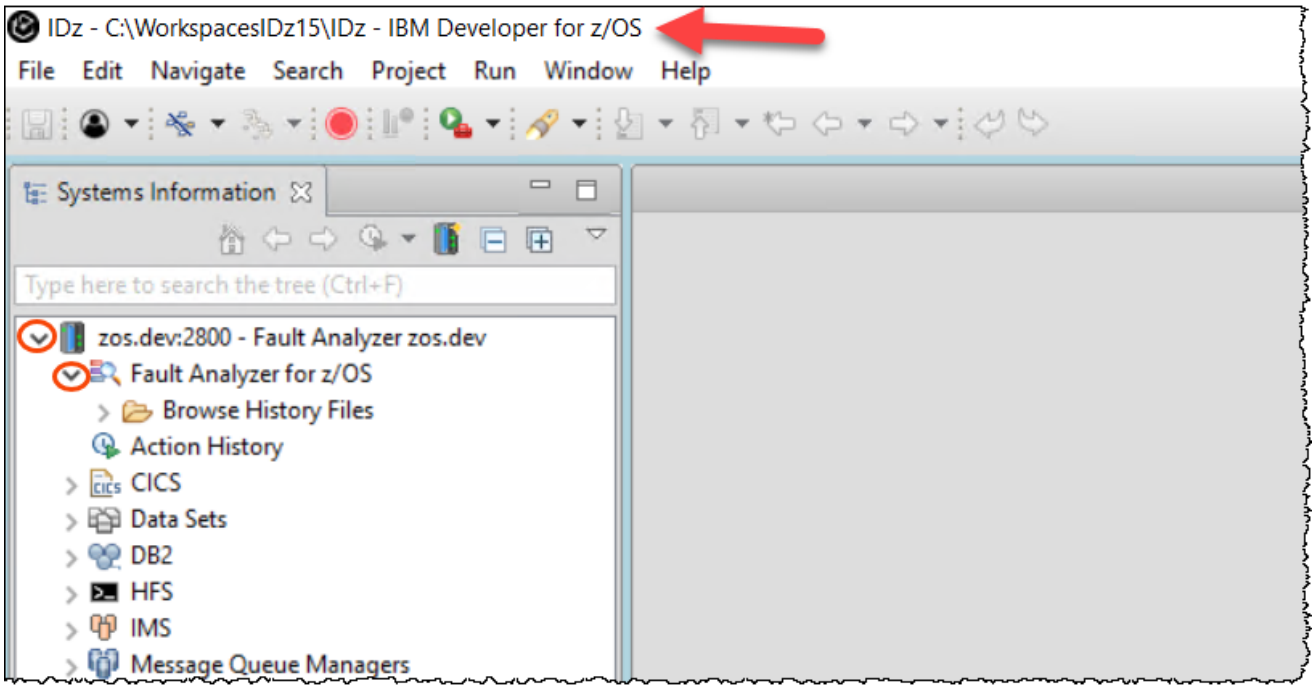


3.1.2 ⏭ Scroll down, select **Fault Analyzer** and click **Open**

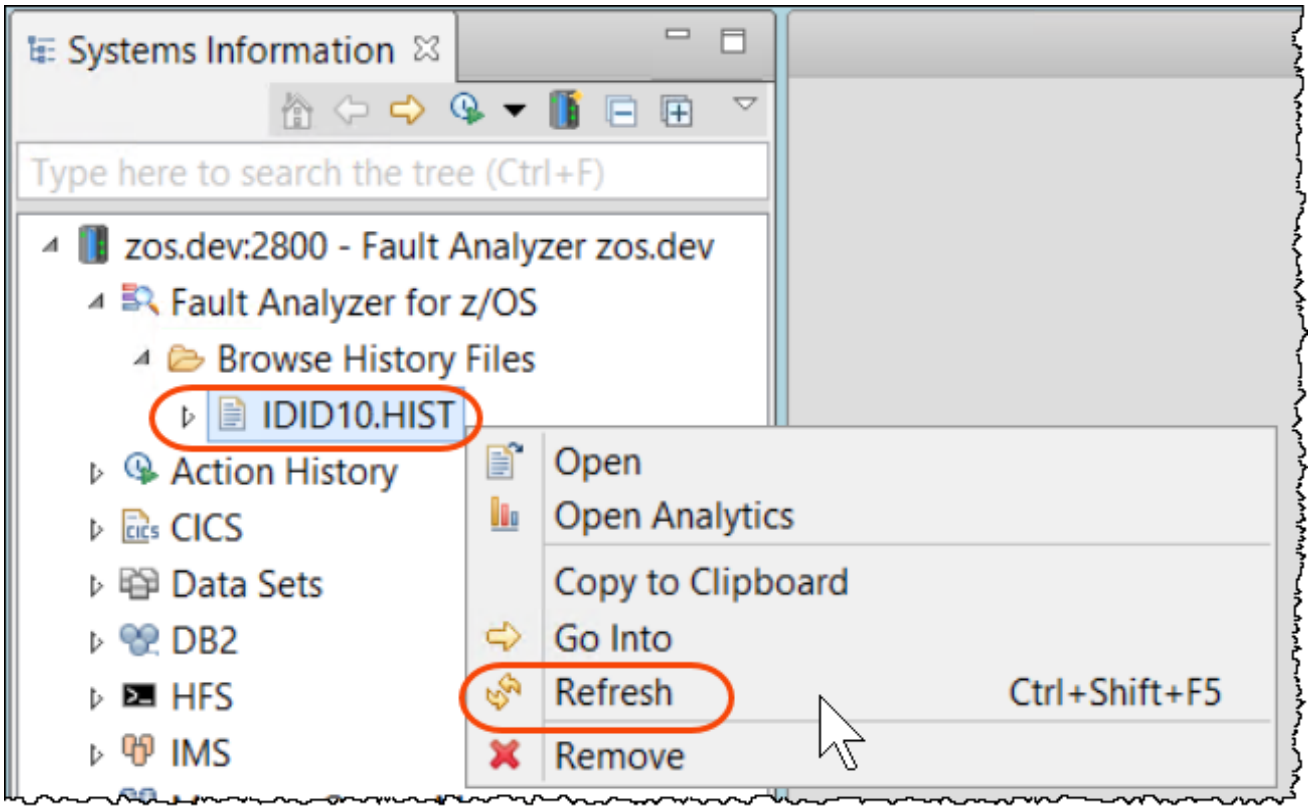


3.1.3 ⏭ Under *Systems Information* view (on left) be sure that **Fault Analyzer for z/OS** is expanded

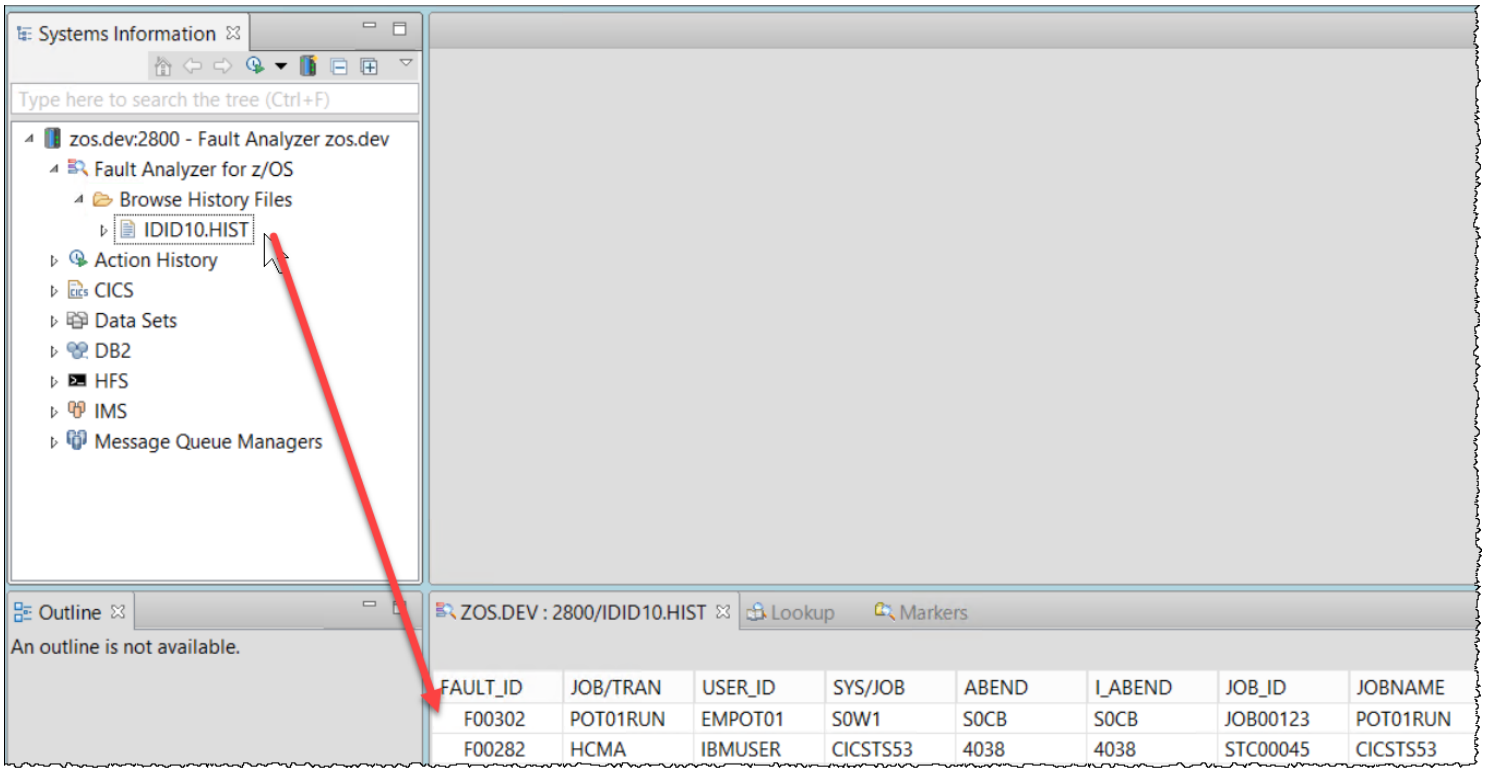
TIP: If you do not have this entry, it is because you are using a wrong IDz workspace. Please contact the instructor.


IDz - C:\WorkspacesIDz15\IDz - IBM Developer for z/OS
File Edit Navigate Search Project Run Window Help
Systems Information
Type here to search the tree (Ctrl+F)
zos.dev:2800 - Fault Analyzer zos.dev
Fault Analyzer for z/OS
Browse History Files
Action History
CICS
Data Sets
DB2
HFS
IMS
Message Queue Managers

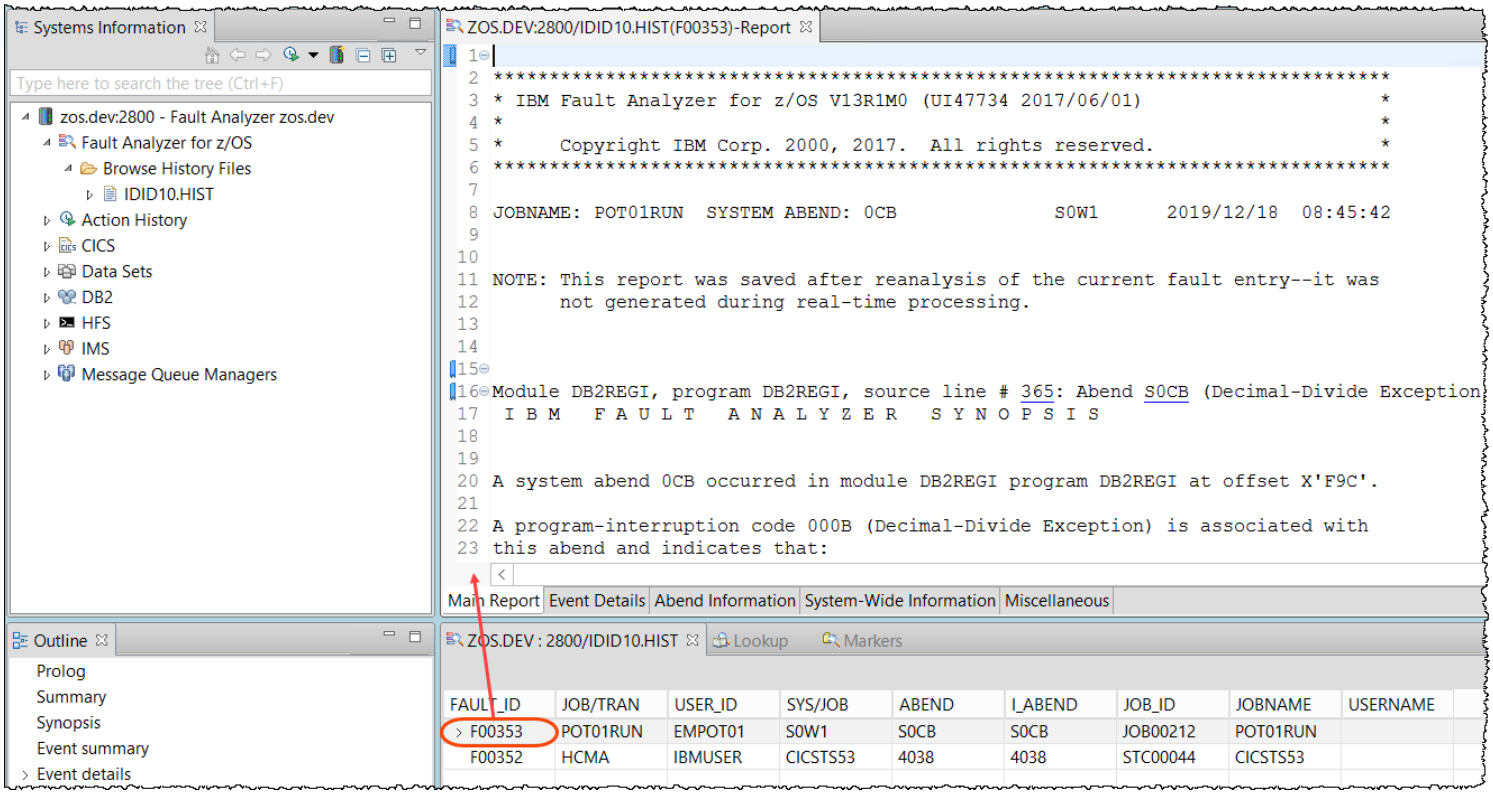
3.1.4 ⏭ Right click **Fault Analyzer for z/OS** and select **Refresh**
⏭ Expand **Browse History File**. You will see the folder ***IDID10.HIST***
⏭ Right click **IDID10.HIST** and select **Refresh** (or **Ctrl + Shift + F5**)



3.1.5 ⏭ If you get a *Sign on* dialog for *Fault Analyzer* use empot01 credentials and click **OK**
You see a list of *FAULT_IDs* on the z/OS system that you are connected to...

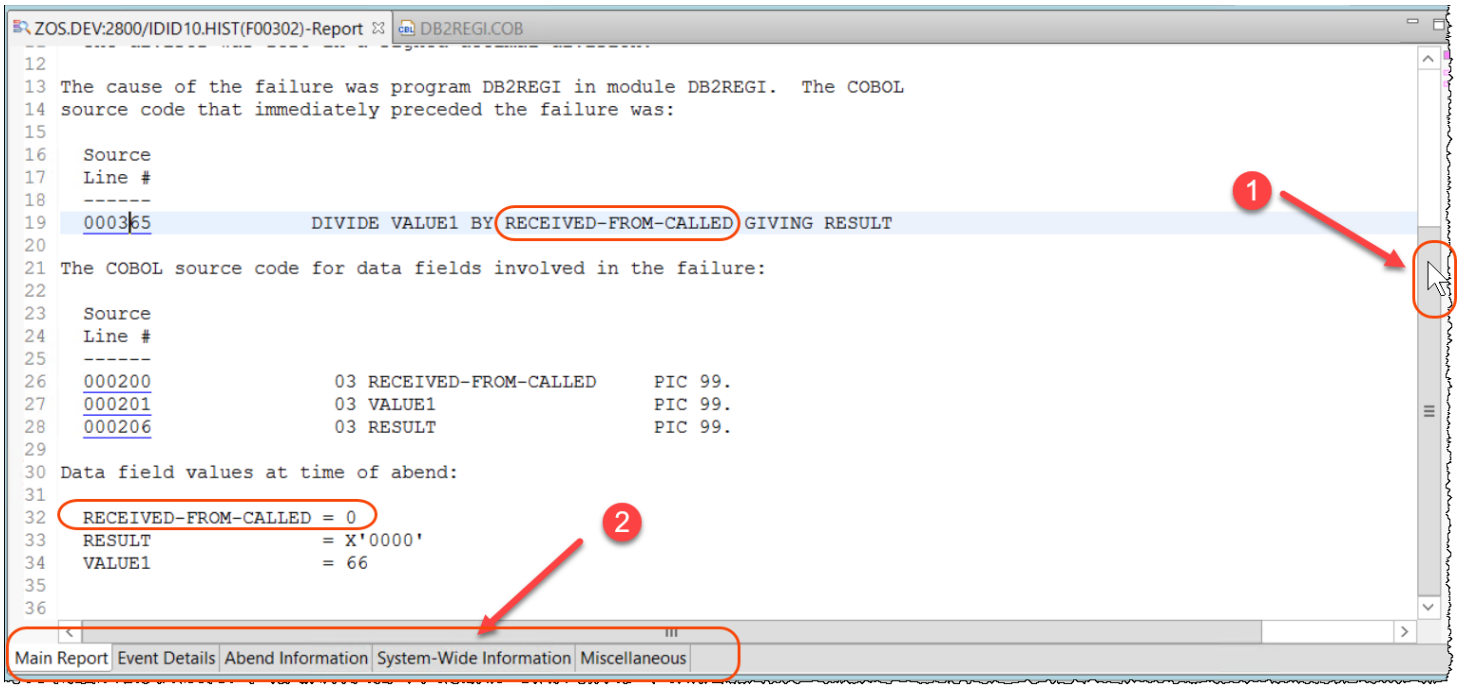

Systems Information
Type here to search the tree (Ctrl+F)
zos.dev:2800 - Fault Analyzer zos.dev
Fault Analyzer for z/OS
Browse History Files
IDID10.HIST
Action History
CICS
Data Sets
DB2
HFS
IMS
Message Queue Managers
Outline
An outline is not available.
ZOS.DEV : 2800/IDID10.HIST
Lookup
Markers
FAULT_ID JOB/TRAN USER_ID SYS/JOB ABEND I_ABEND JOB_ID JOBNAME
F00302 POT01RUN EMPOT01 S0W1 S0CB S0CB JOB00123 POT01RUN
F00282 HCMA IBMUSER CICSTS53 4038 4038 STC00045 CICSTS53

3.1.6 ⏭ On the ***ZOS.DEV:2800/IDID10.HIST*** view locate the latest job name **POT01RUN** and **double click** on it.
You will see the report being downloaded from the z/OS to your Windows client. The report below will be displayed

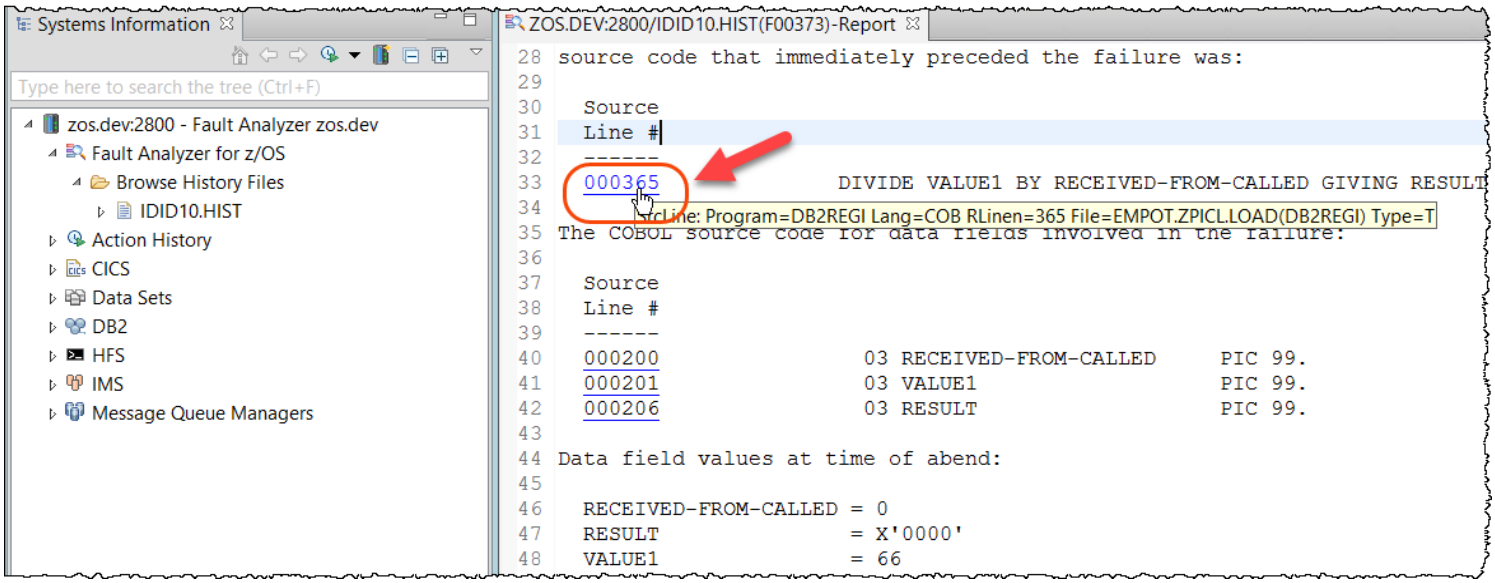


3.1.7 ⏭ Scroll the report down and you will see that the field **RECEIVED-FROM-CALLED** used on the division has “0”. So the abend is because of a divide by zero.

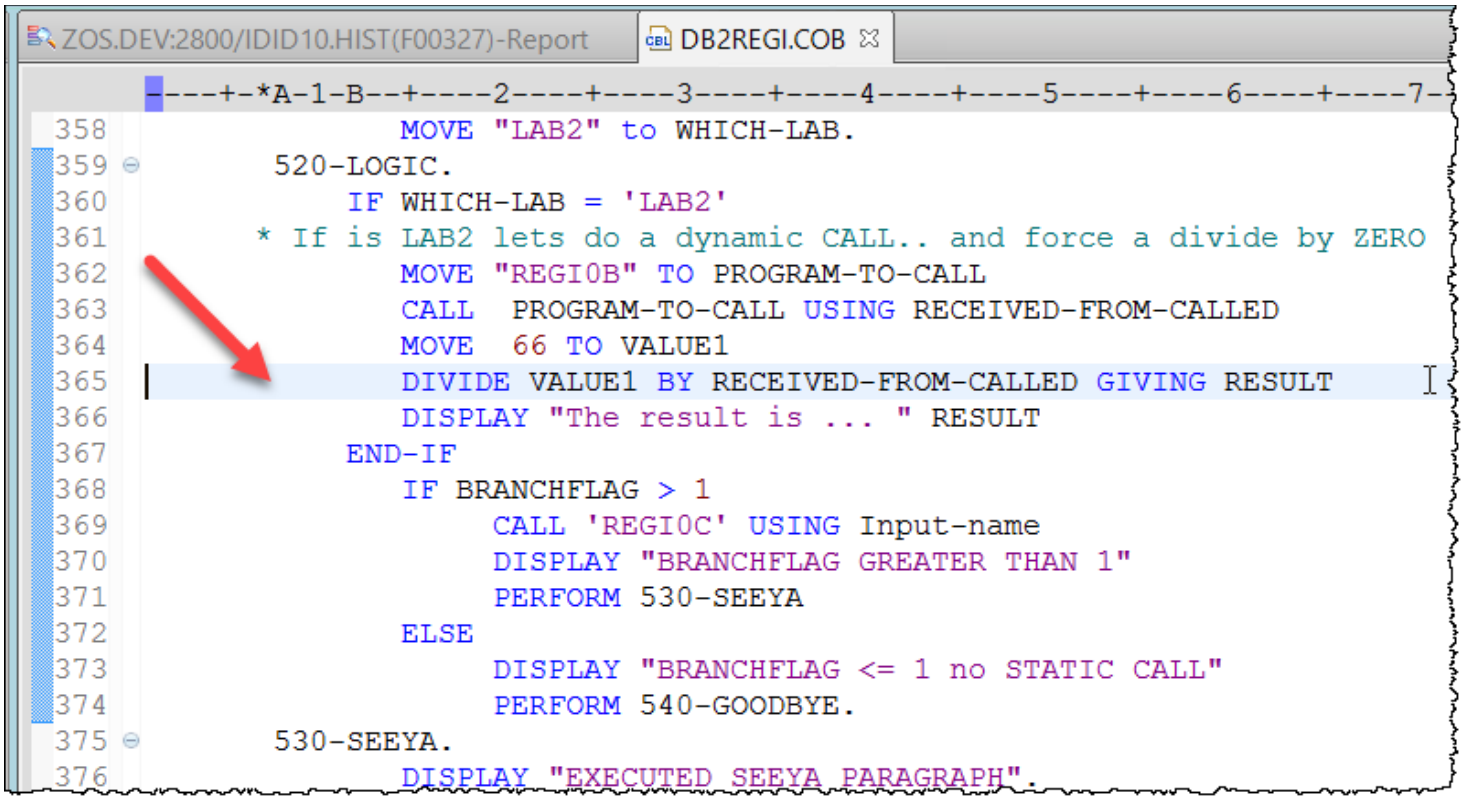
⏭ Also notice that there are other tabs on this panel (like Event Details, Abend Information, etc.).
You may try those tabs to get more information about the abend.



3.1.8 Using the *Main Report* view Click on the link **000365**
This will show the COBOL statement that caused the *0CB abend*

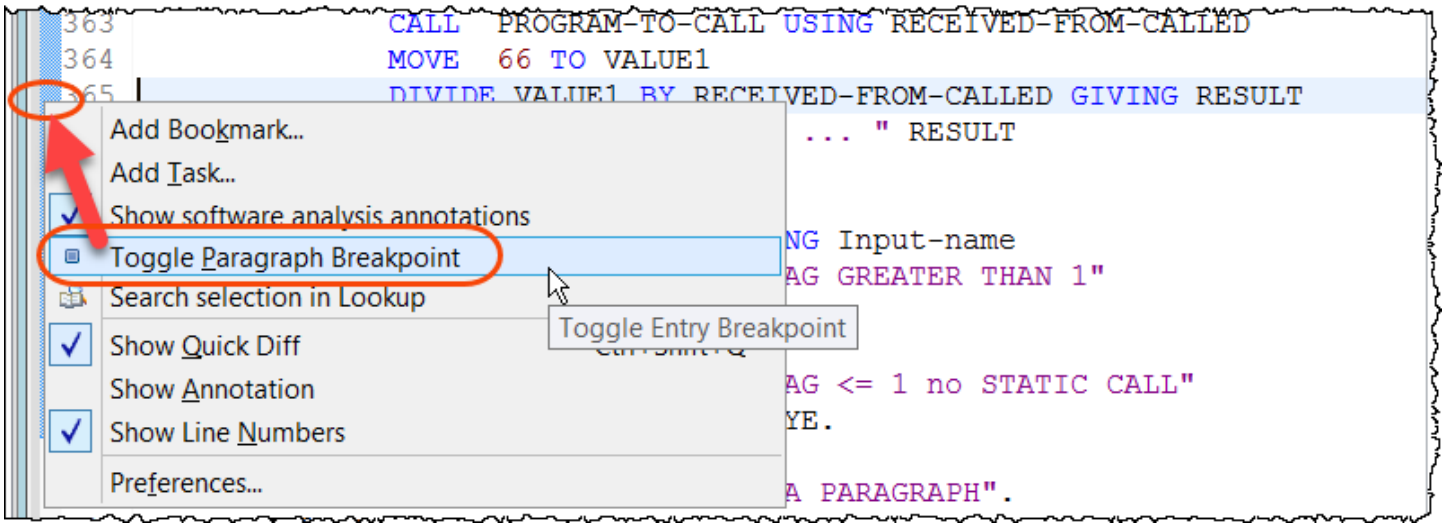


3.1.9 The program editor shows the line with the statement that is causing the abend.
Notice that this is NOT the COBOL source code. This what is on the “minidump” downloaded by Fault Analyzer.
There is no sense to make changes here. But you will see what caused the abend.

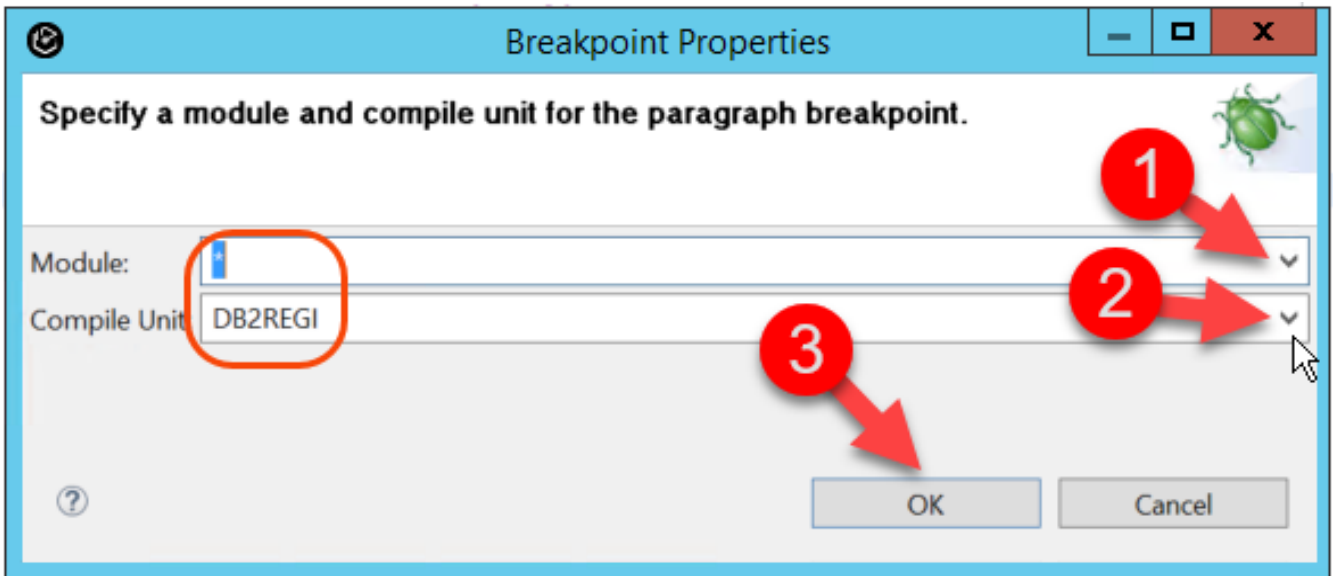


Tip: If the font is not clear on the editor, it is because we increased the font to 150%. To make this smaller start the **Control Panel** > **Display** > **set a custom scaling level** > reduce to **125%** (we did set to 150%).

3.1.10 On line **365**, right-click in the ruler area (the blue line on left) and select **Toggle Paragraph Breakpoint**. (also known as Toggle Entry Breakpoint)



3.1.11 On the Breakpoint Properties dialog use drop down to select *, and select or type **DB2REGI** as Compile Unit and click **OK**



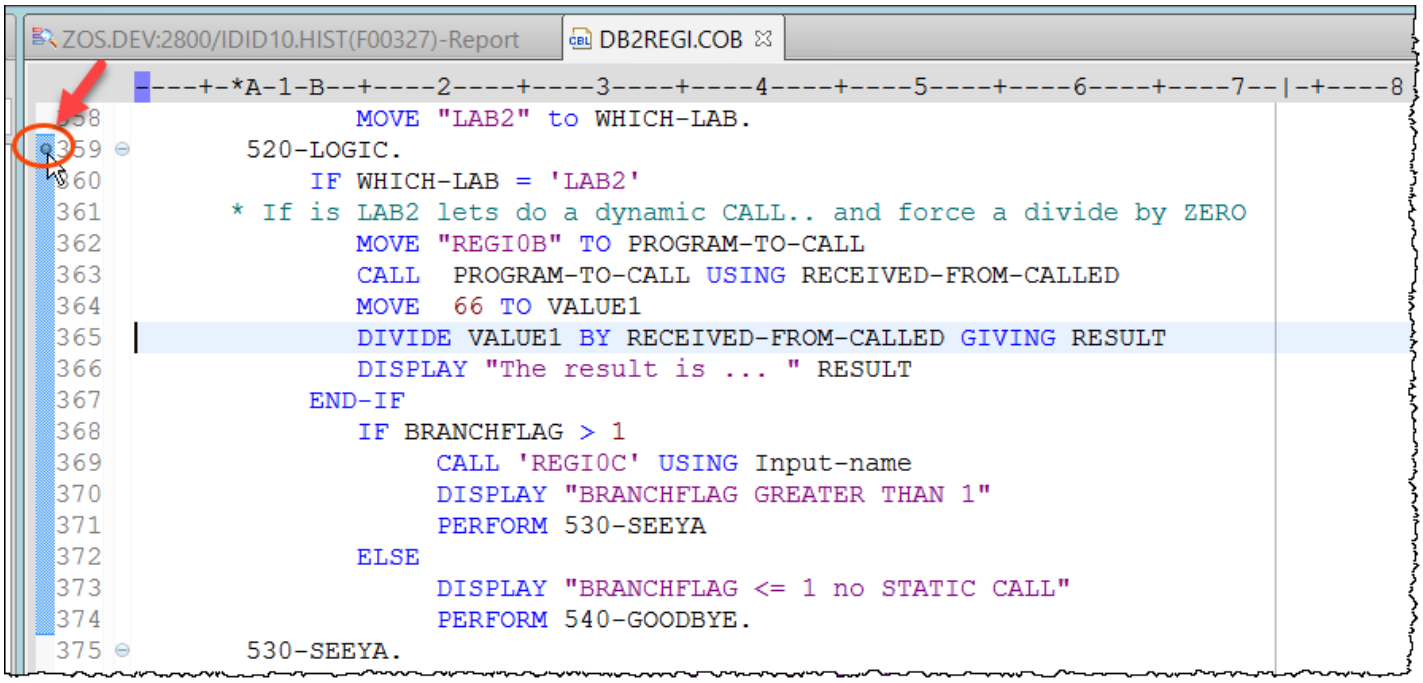
What is *Toggle Paragraph Breakpoint*?

Toggle Paragraph Breakpoint provides the ability to set paragraph breakpoints prior to starting a debug session. Because these breakpoints are set using the original source files, they persist between debug sessions.

On previous versions, breakpoints could only be set at the level of the generated program listing files.

This allows a more natural edit, compile and debug workflow on the original source files.

3.1.12 **Scroll up** a bit. The *Toggle Paragraph Breakpoint* is set at the **520-LOGIC** paragraph. This break point may optionally be used when debugging the COBOL code.



3.1.13 Close all opened editors using **CTRL+ Shift + F4**.
Or just click on the . of each opened editor.

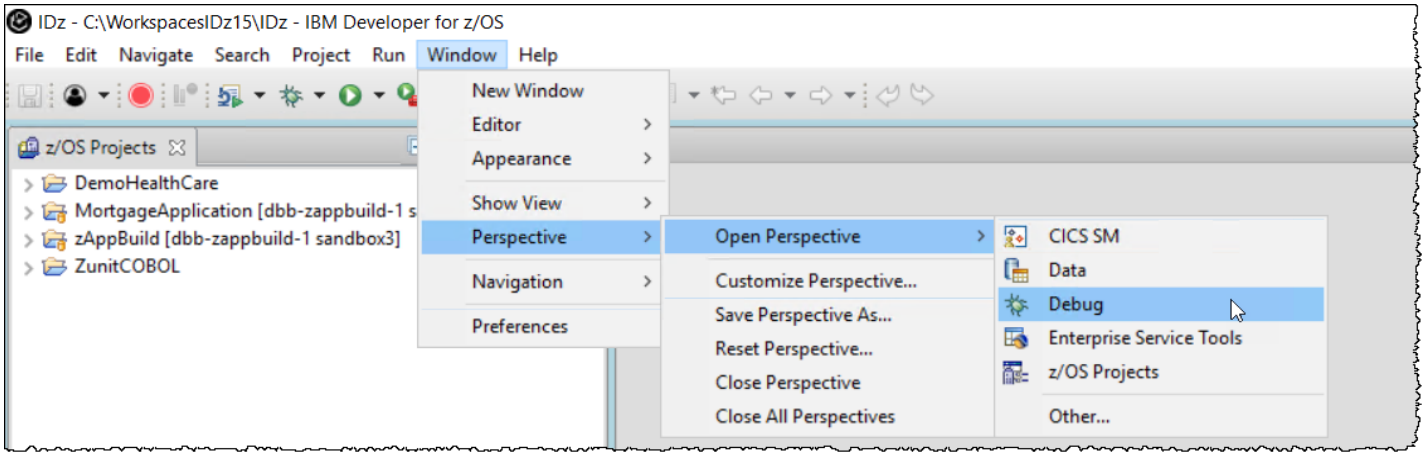
Section 4. Using the IBM z/OS Debugger for a temporary fix

You will use the Debug to verify the ABEND and make a runtime fix.

___ 4.0 Be sure that IDz client is listening on port 8001

Usually this step is NOT required, but in our cloud environment we have timeouts that force ports to be closed.

4.0.1 Go to the *Debug Perspective* by selecting **Windows > Perspective > Open perspective > Debug**



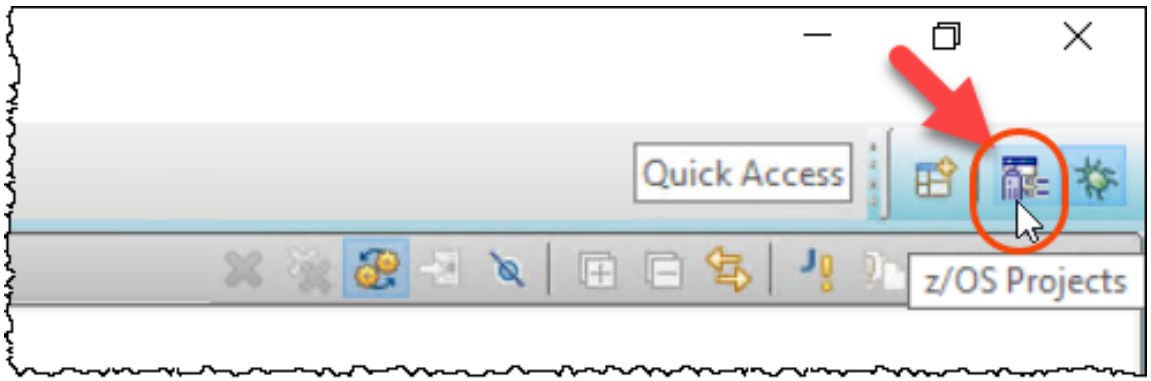
4.0.2 If the icon is **red**, click and select **Start listening on port 8001**

4.0.3 The listening icon will turn green and the IDz client is listening on port 8001.

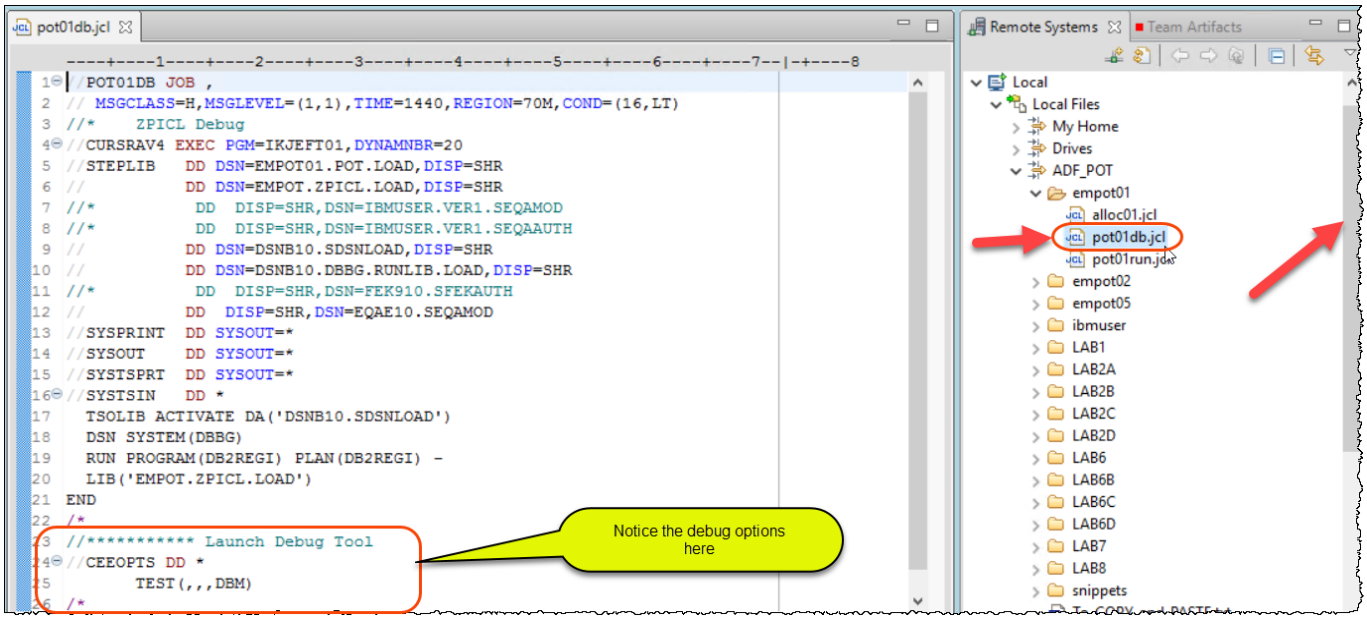
___ 4.1 Submit the JCL to invoke the Debug

You can now submit the JCL to execute again with the Debug option.

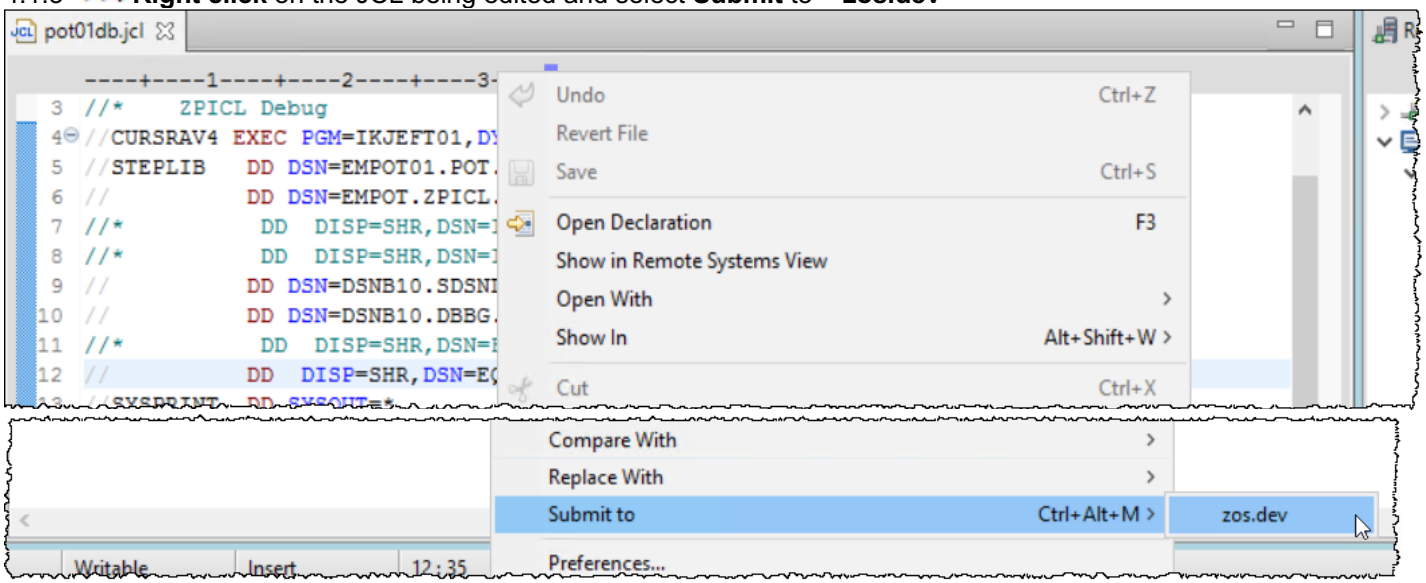
4.1.1 Using the top right corner Go to the **z/OS Projects** Perspective by clicking on the icon below.



4.1.2 Using *Remote System View*, **scroll up** to locate the file **pot01db.jcl** under **Local/Local Files/ADF_POT/empot01** and **double click** to edit..



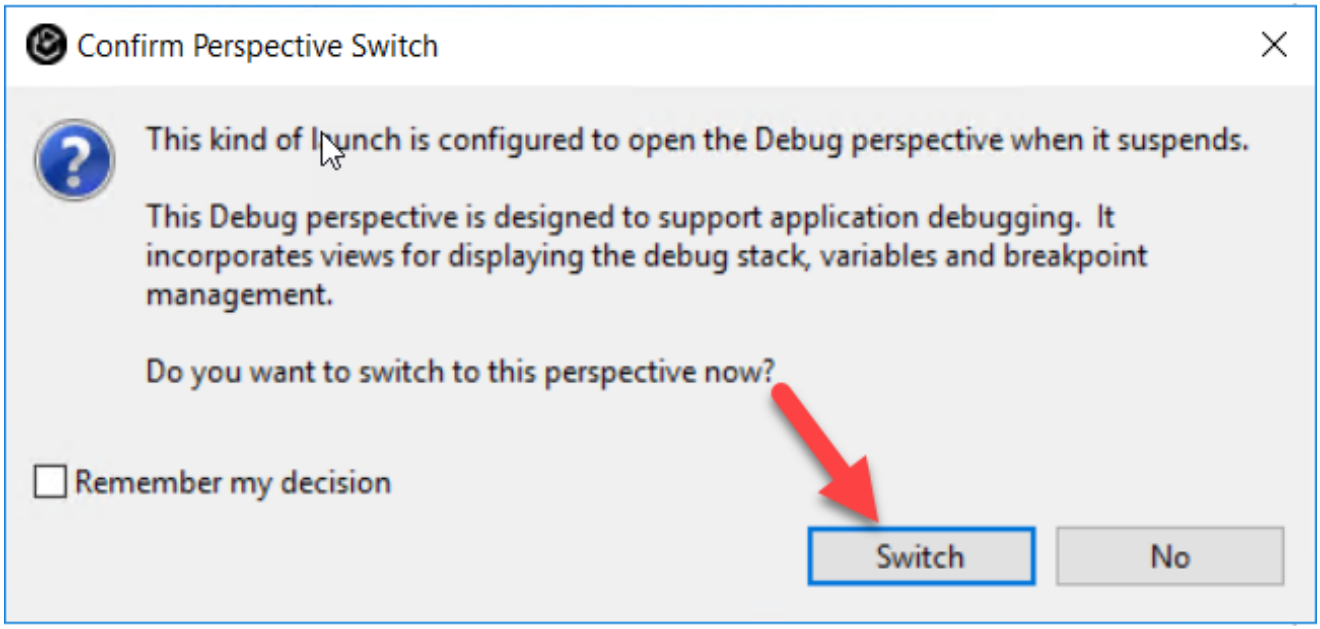
4.1.3 **Right click** on the JCL being edited and select **Submit** to > **zos.dev**



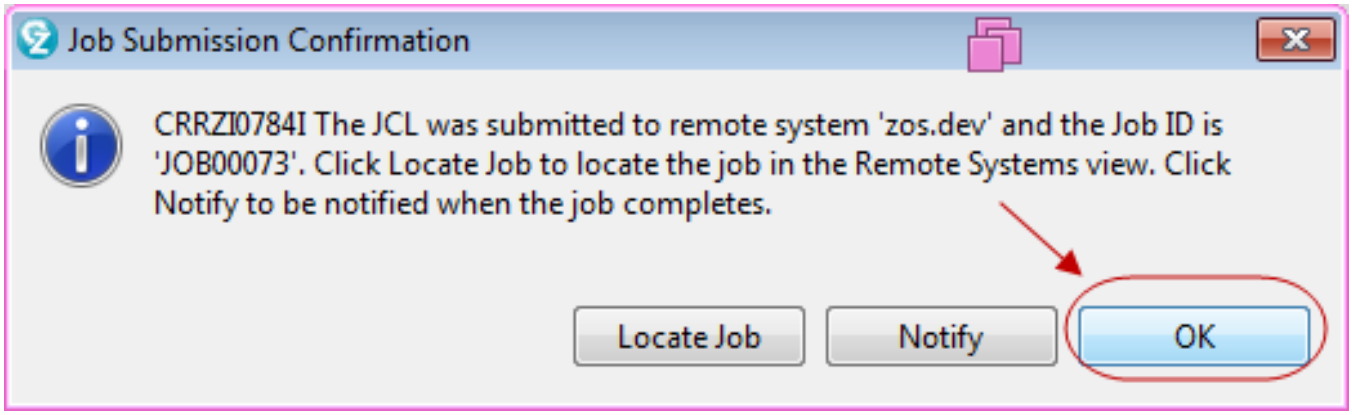
4.1.4 Once the job starts the z/OS debug will "talk" with you. Notice that the communication will be via the IDz connection (RSE), you don't need to specify IP addresses. This may work even when firewalls are in place.

Click **Switch** to open the *Debug* Perspective

Notice: Depending how fast are you the dialogs order here could be different.

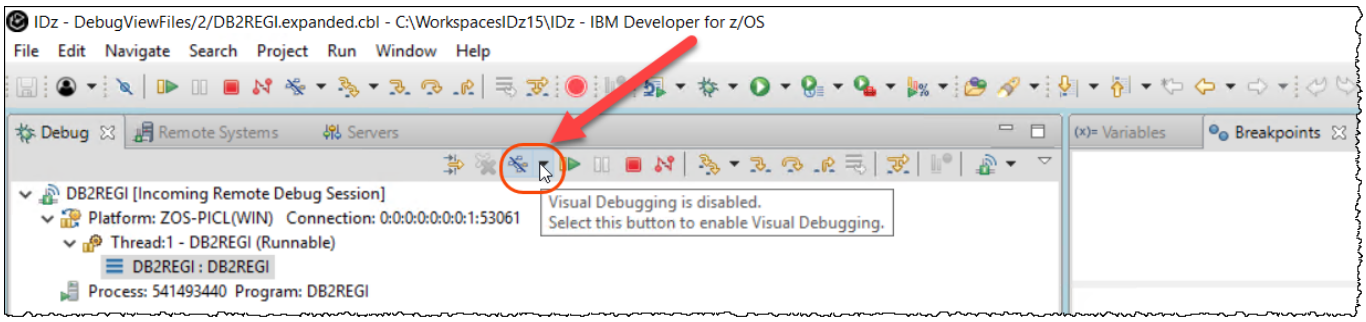

Confirm Perspective Switch
This kind of launch is configured to open the Debug perspective when it suspends.
This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.
Do you want to switch to this perspective now?
Remember my decision
Switch
No

4.1.5 Also click **OK** for this dialog below



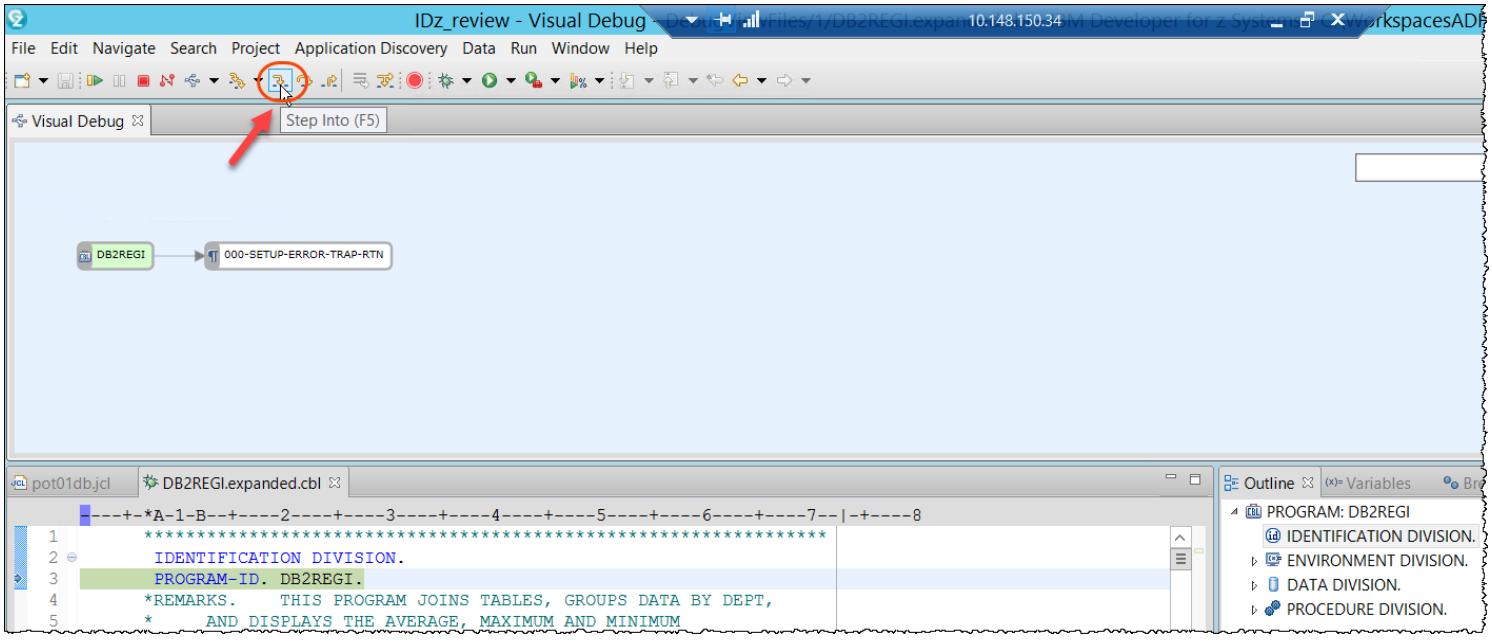
4.1.6 Using the *Debug* perspective:

Click the icon  to enable Visual Debugging which shows the ***Visual Debug*** view.
If a dialog pops up click **YES**.

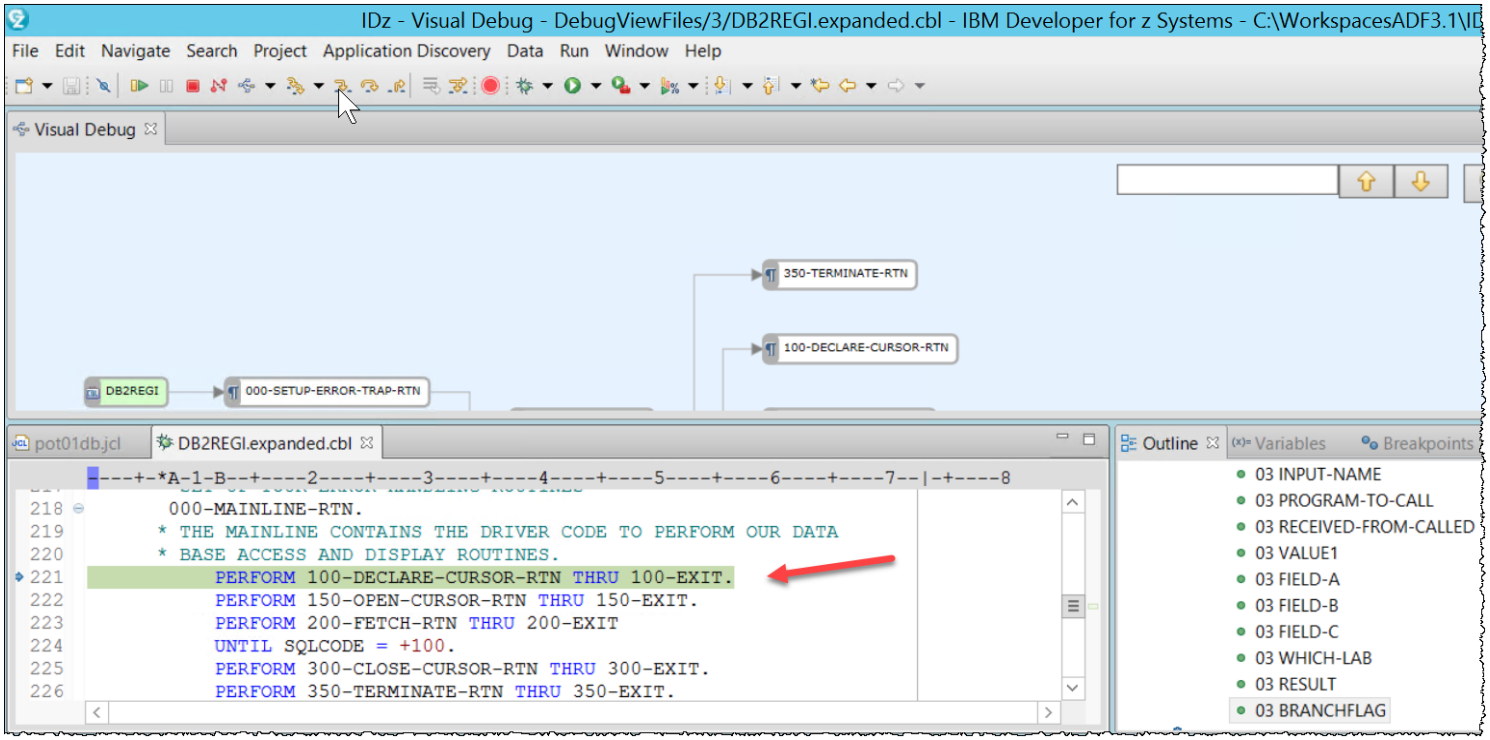


4.1.7 The *Visual Debug* view show the paragraphs being executed (in the top)

Click the icon  or **F5** (*Step into*) to execute first line of code.

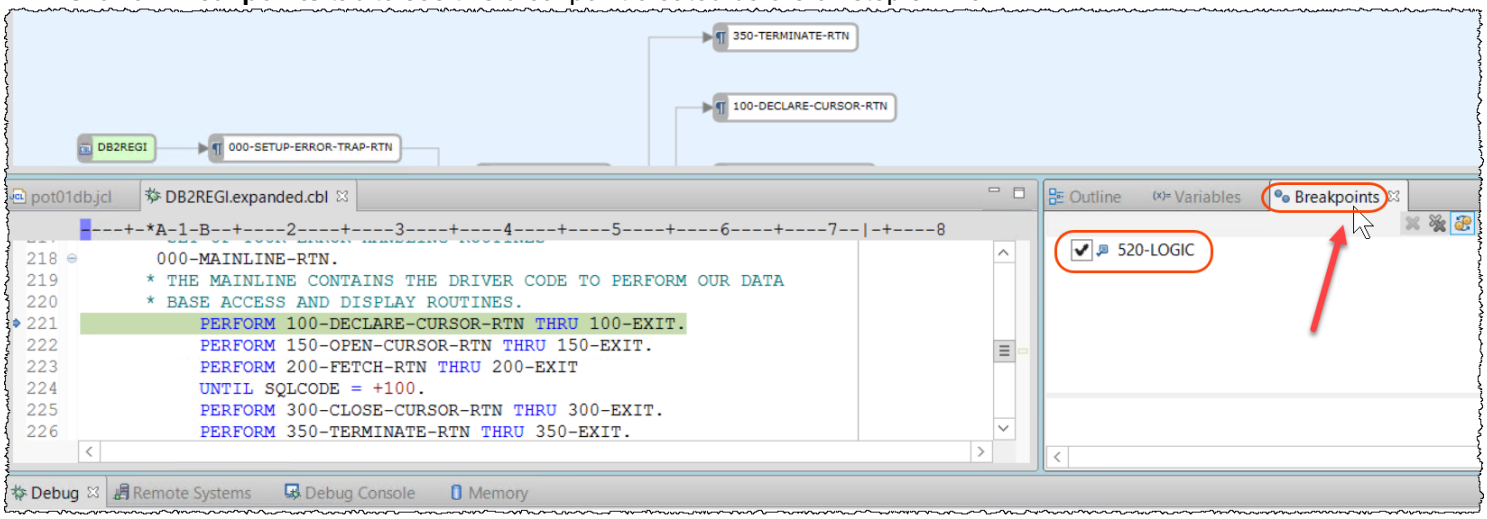


4.1.8 The execution will stop at the statement that will execute
`PERFORM 100-DECLARE-CURSOR-RTN THRU 100-EXIT`



4.1.9 On the step 3.1.10 when using the *Fault Analyzer,* you created a *paragraph breakpoint* even without having the execution started. This is handy since it will show the area that needs to be debugged..

⏭ Click on **Breakpoints** tab to see this breakpoint created before on step 3.1.10.

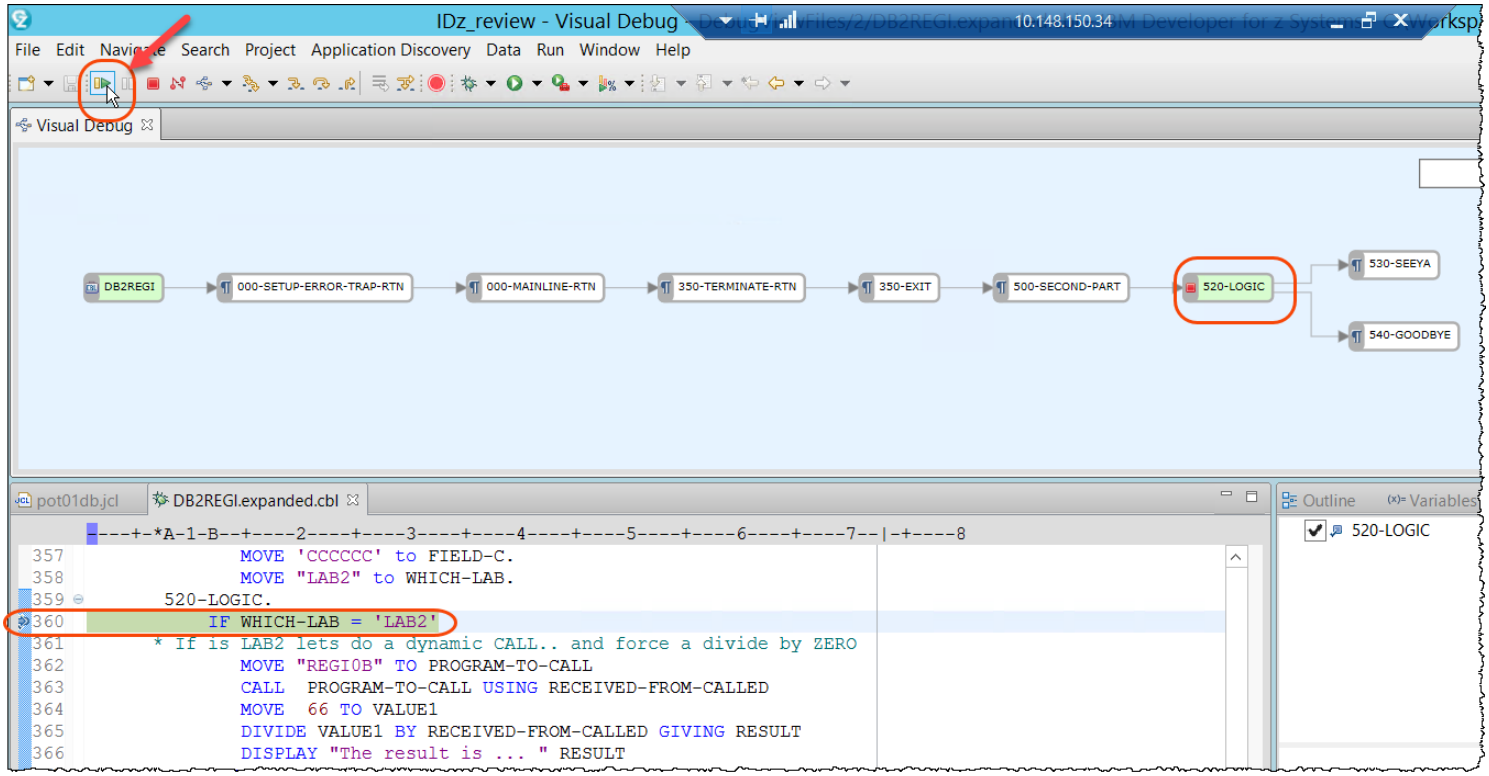


What is the Visual Debugging?

Visual debugging allows you to interact with your COBOL or PL/I debug session using the program control flow diagram.
With this diagram, you can visualize the stack trace, set breakpoints, and run to a selected

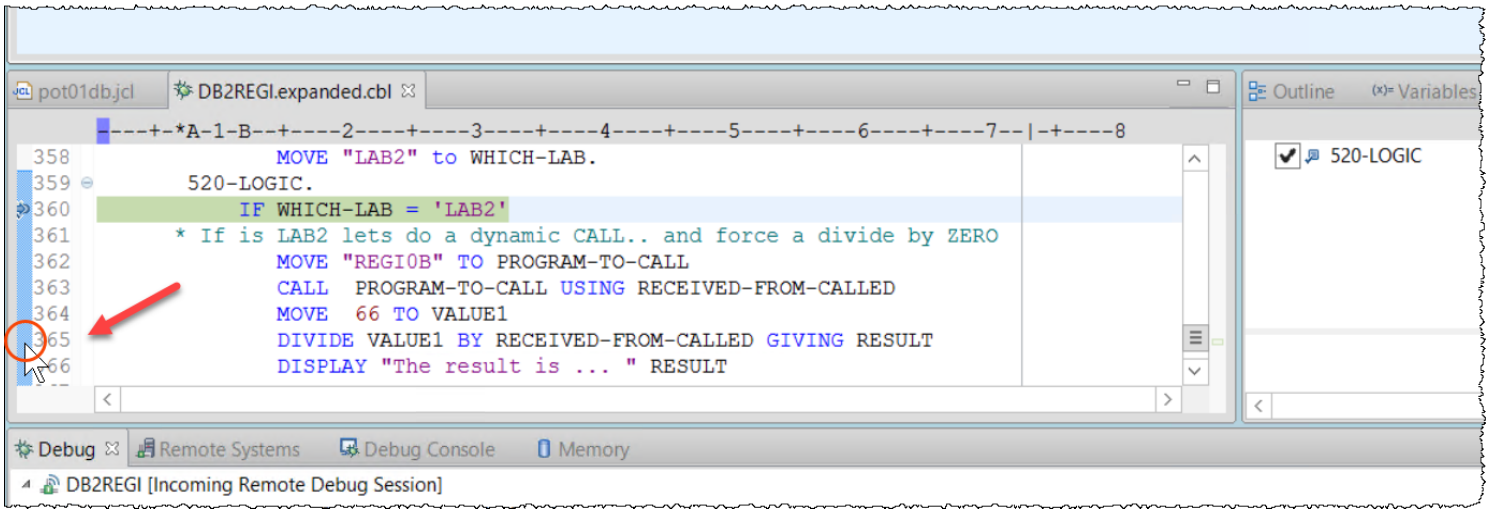
call path. In COBOL, the stack trace represents the paragraph call chain.
In PL/I, the stack trace represents the procedure call chain.

4.1.10 Click on or press **F8** and notice that the execution will stop on line 360 since a *Paragraph Entry Breakpoint* was created before. This line is about to be executed.

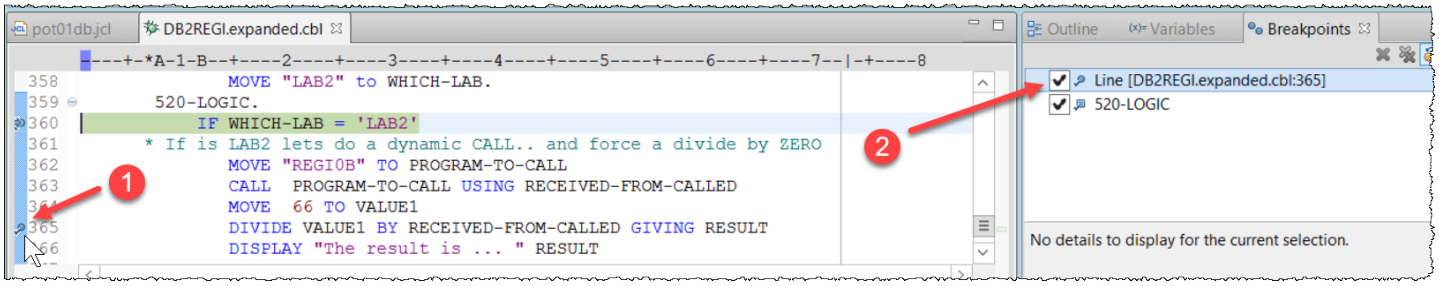


4.1.11. As you verified using Fault Analyzer, the abend occurred on line 365 where you had a divide by zero (see step 3.1.10). Now you will add a breakpoint on this line and change the values to avoid the abend.

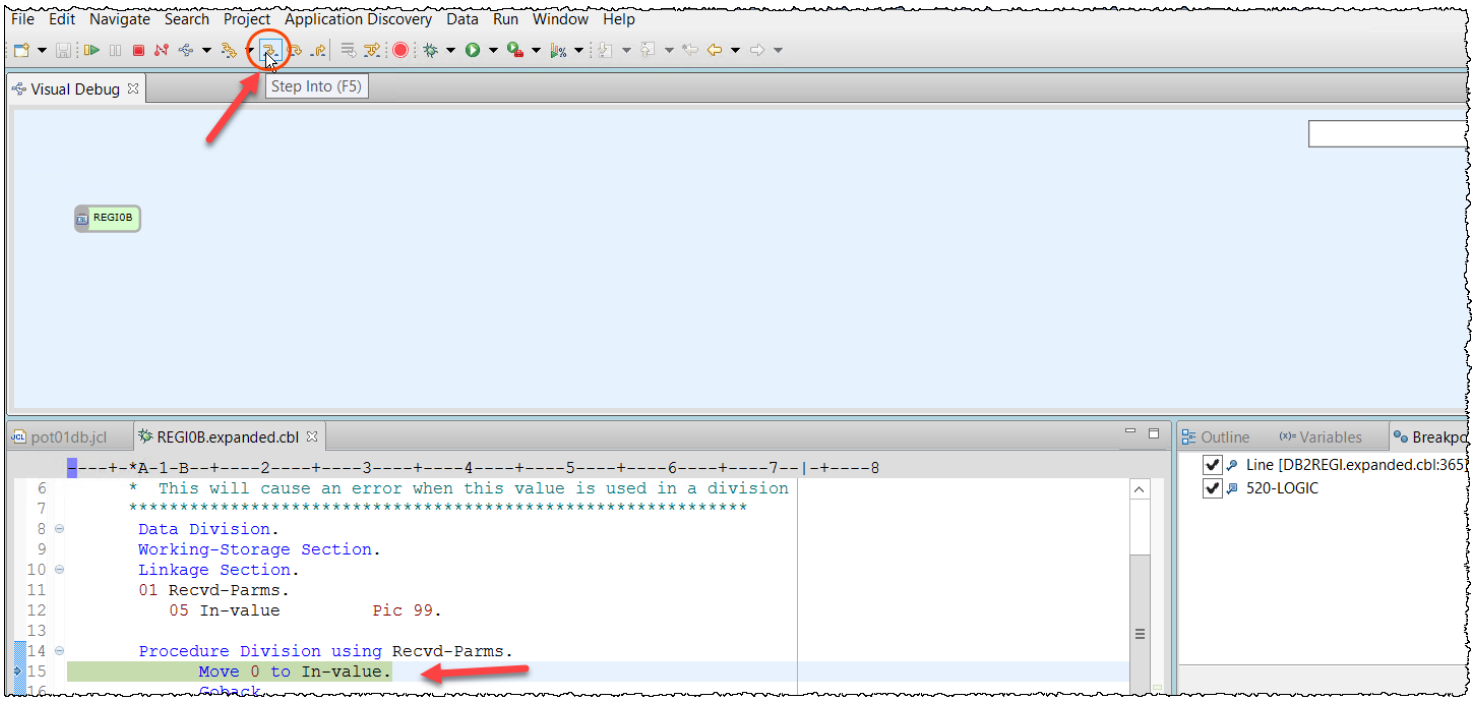
On the COBOL editor move the mouse to the **blue column** on left and **double click** on the line 365 `DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT` to create a breakpoint.



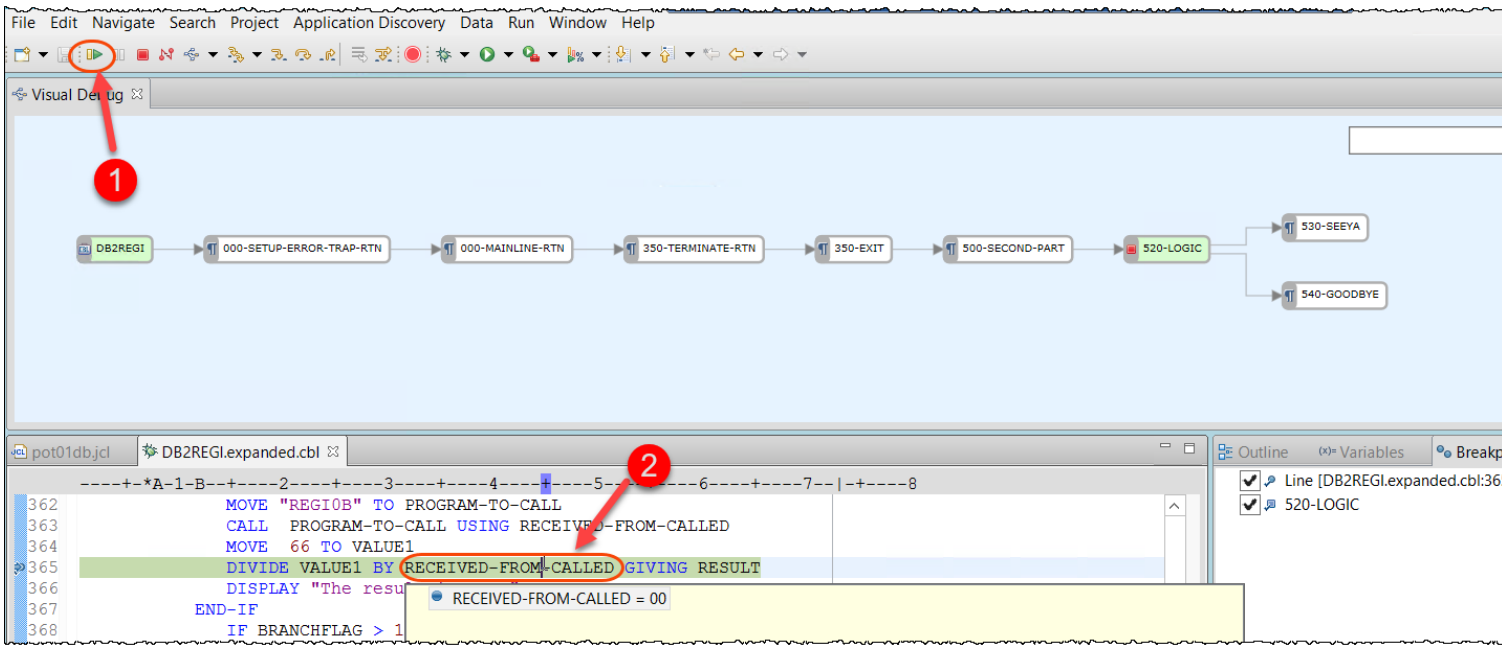
4.1.12 Notice that a small circle is shown on the left of line **365** and also a breakpoint is displayed on the Breakpoint view



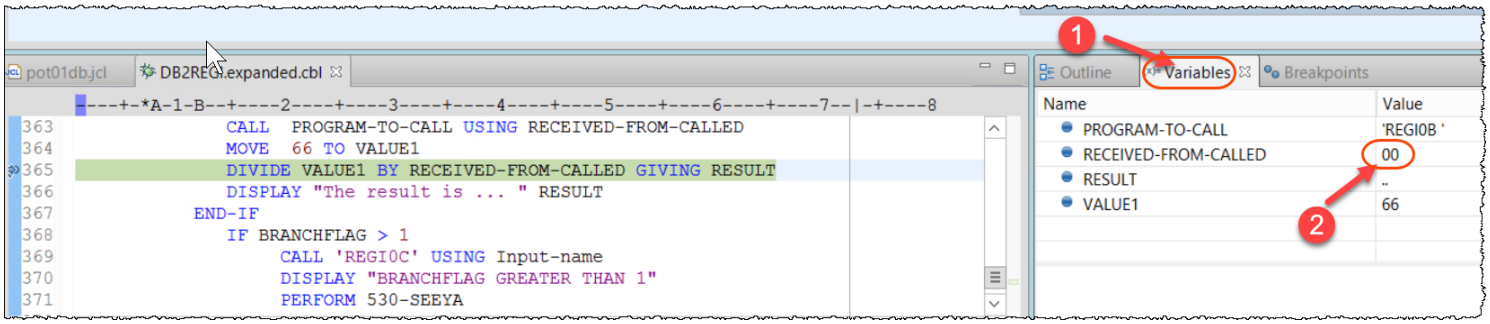
4.1.13 Continue by clicking on or using **F5** until you will see that a program named **REGI0B** is called and this program is returning a value of 0.



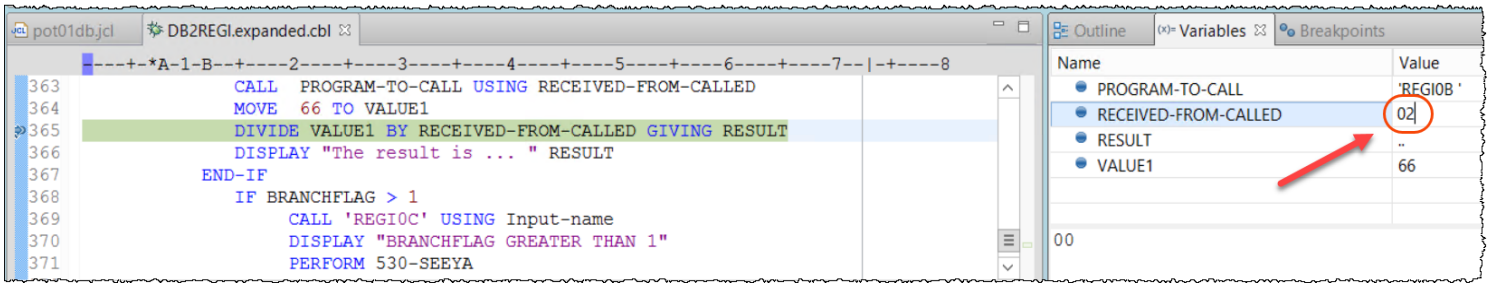
4.1.14  Click on  or press **F8.**
The execution will stop at the breakpoint that you created (line 365). .
 Move the cursor to **RECEIVED-FROM-CALLED** variable and click to see the **00** value. .
(Tip: under the cloud instance the behavior may act different and you may need to click on the editor area before moving the mouse to that field).



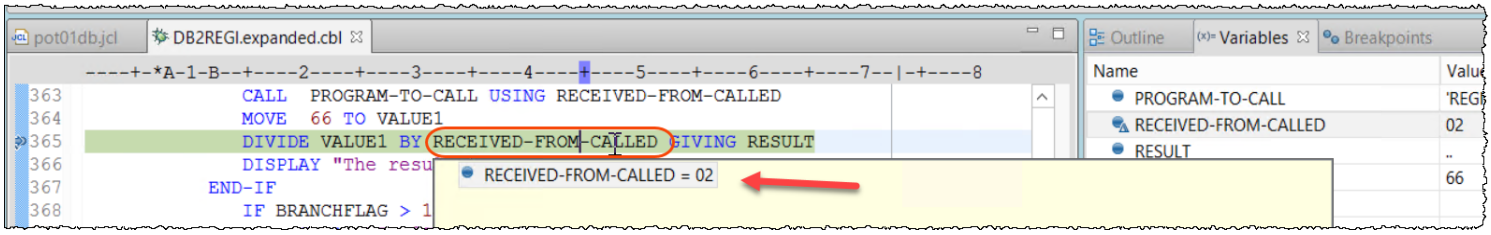
4.1.15  Click the **Variables** tab (right) and  verify that **RECEIVED-FROM-CALLED** is **00**.



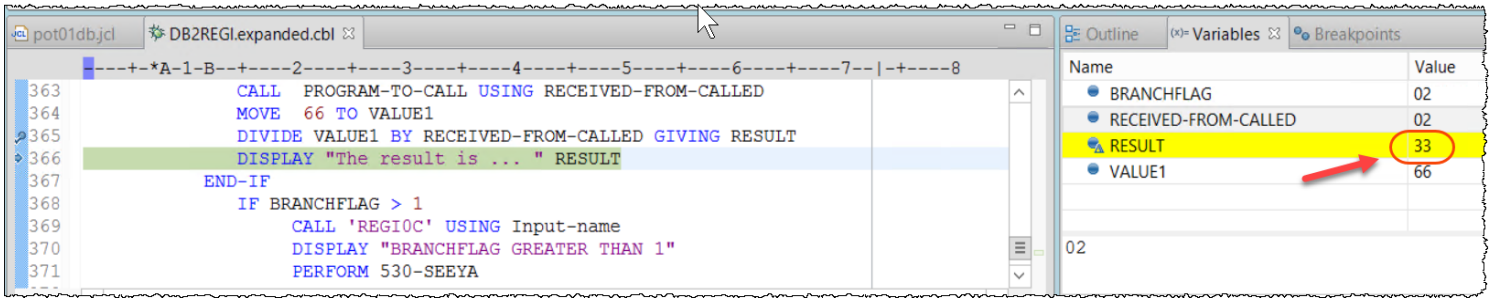
4.1.16  This is the abend cause. You must change the value to something different than 0.
 Click on **RECEIVED-FROM-CALLED** value of **00** and modify to **2**, just overtyping and press **enter.**


pot01db.jcl
DB2REGI.expanded.cbl
Outline
Variables
Breakpoints
Name
Value
PROGRAM-TO-CALL
'REGI0B '
RECEIVED-FROM-CALLED
02
RESULT
..
VALUE1
66
00
CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
MOVE 66 TO VALUE1
DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
DISPLAY "The result is ... " RESULT
END-IF
IF BRANCHFLAG > 1
CALL 'REGI0C' USING Input-name
DISPLAY "BRANCHFLAG GREATER THAN 1"
PERFORM 530-SEEYA

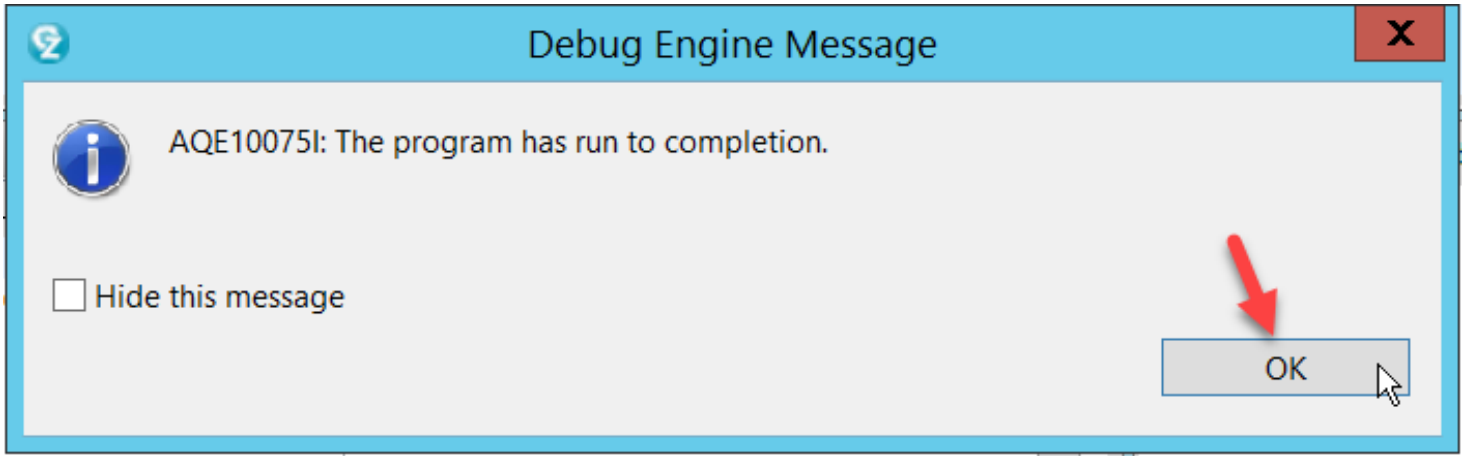
4.1.17 Again, move the cursor to **RECEIVED-FROM-CALLED** variable and now see the value **02**.
(Tip: under the cloud instance the behavior may act different and you may need to click on the editor area before moving the mouse to that field).



4.1.18 Click the icon (Step into) or press **F5** to see the next line being executed
This time the divide by 2 give you a result of **33.**



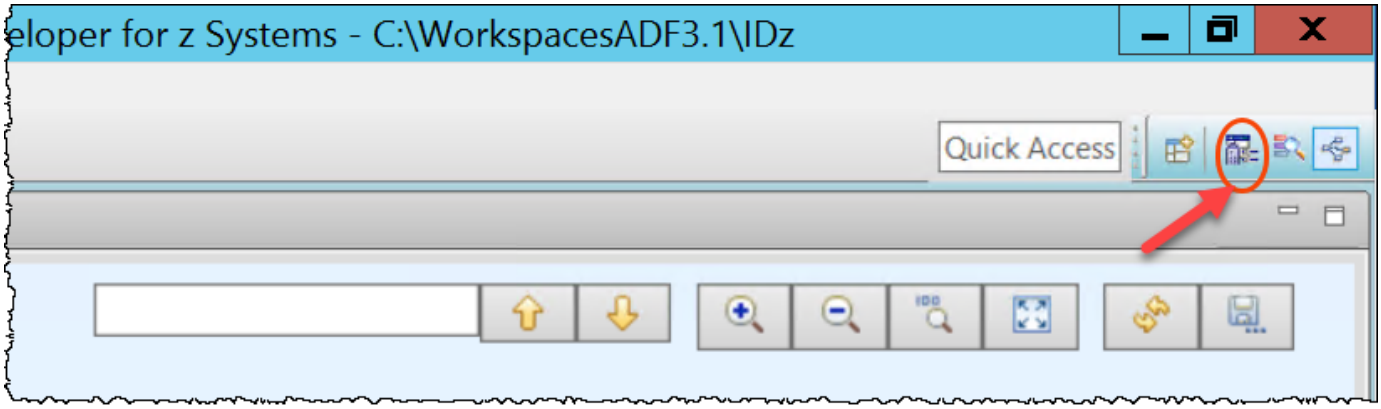
4.1.19 Click on **(or F5)** few times
and **Resume** (**F8**) when you are satisfied so the program will execute until the end.



4.1.20 Click **OK** to close the dialog above.

To fix this bug definitely you will modify the program **REGI00B** that is returning 0 to be used in a division.
You will do that later. For now, optionally you can play again with the Debugger and use the Visual Debugger.
Or continue after the optional steps.

4.1.21 Go back to the **z/OS Projects Perspective** by clicking on the icon the top right corner.



___ 4.2 (OPTIONAL) Using the Visual Debugger for Stack pattern breakpoints

If you are not running late and interested on this subject, execute the steps below, **otherwise jump to 4.3**

Stack pattern breakpoint

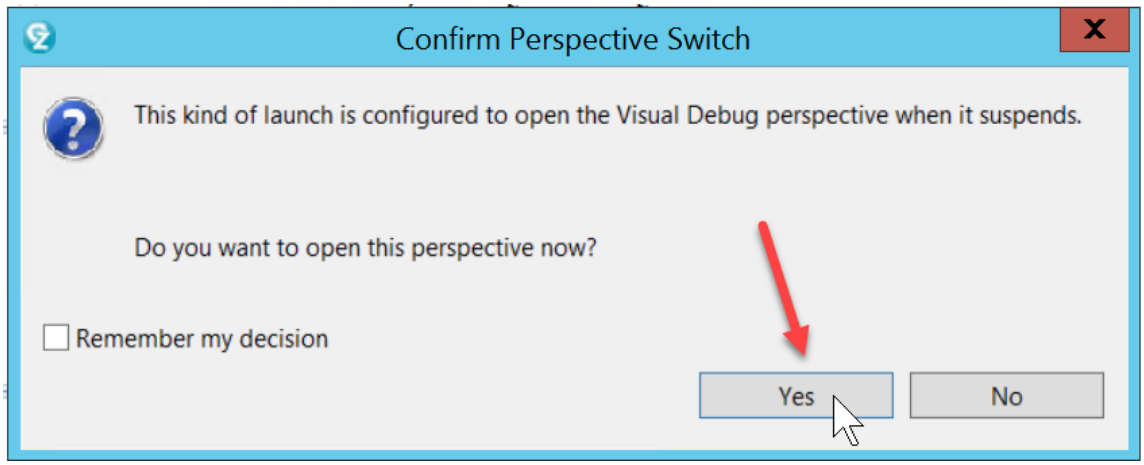
A stack pattern breakpoint is a special type of conditional breakpoint. With this feature, you can specify that you only want to stop at a location when the current stack trace contains a predefined stack pattern.

Visual debugging supports a stack pattern integration feature, which allows you to select a connected path from the program control flow diagram and set a stack pattern breakpoint using the selected path as a stack pattern.

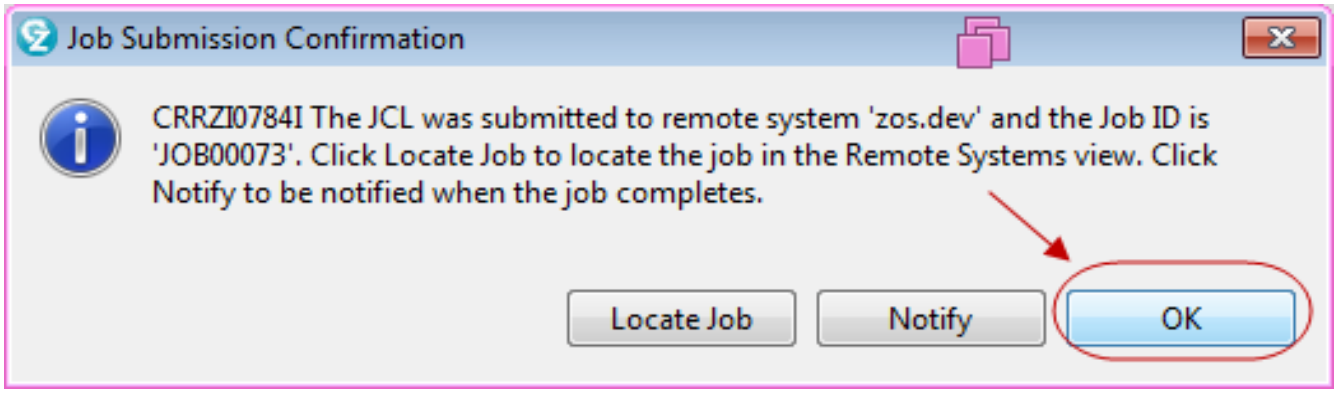
4.2.1 (Optional) **Right click** on the JCL being edited and select **Submit** > **zos.dev**

4.2.2 (Optional) Once the job starts the z/OS debug will “talk” with you.

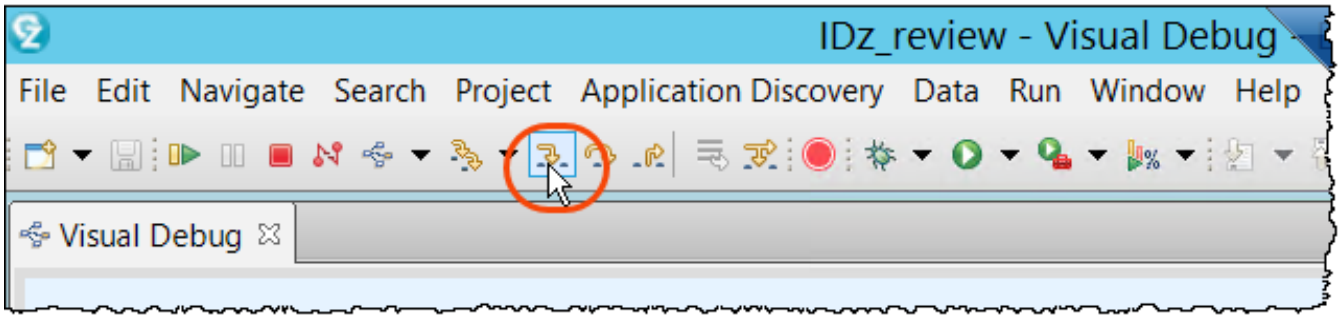
⏭ Click **Yes** to open the *Visual Debug* Perspective
(depending on the z/OS the order here could be the next step first).



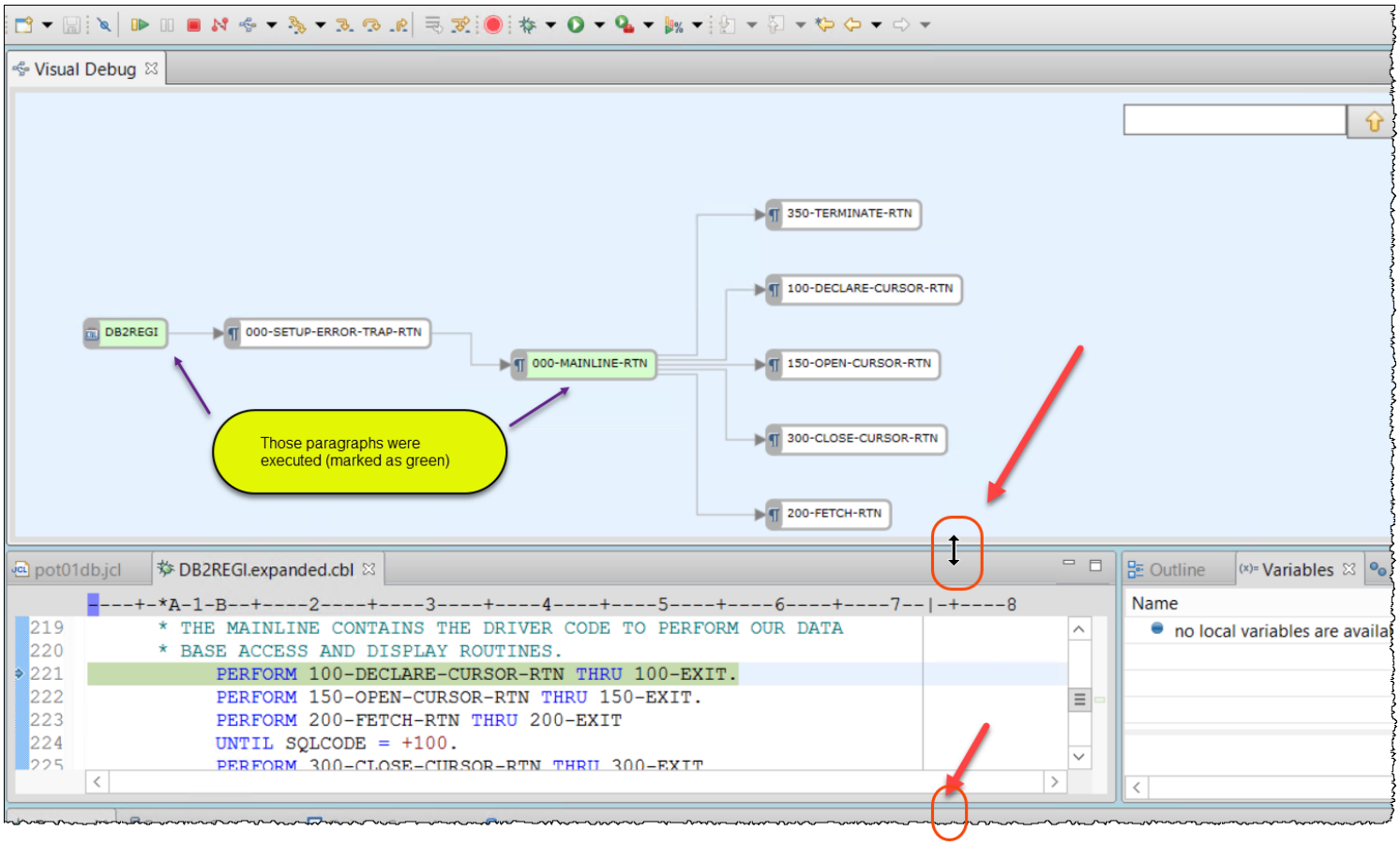
4.2.3 (Optional) Also click **OK** for this dialog below



4.2.4 (Optional) Using the *Visual Debug* perspective:
Click on **icon** (or Press **F5**)



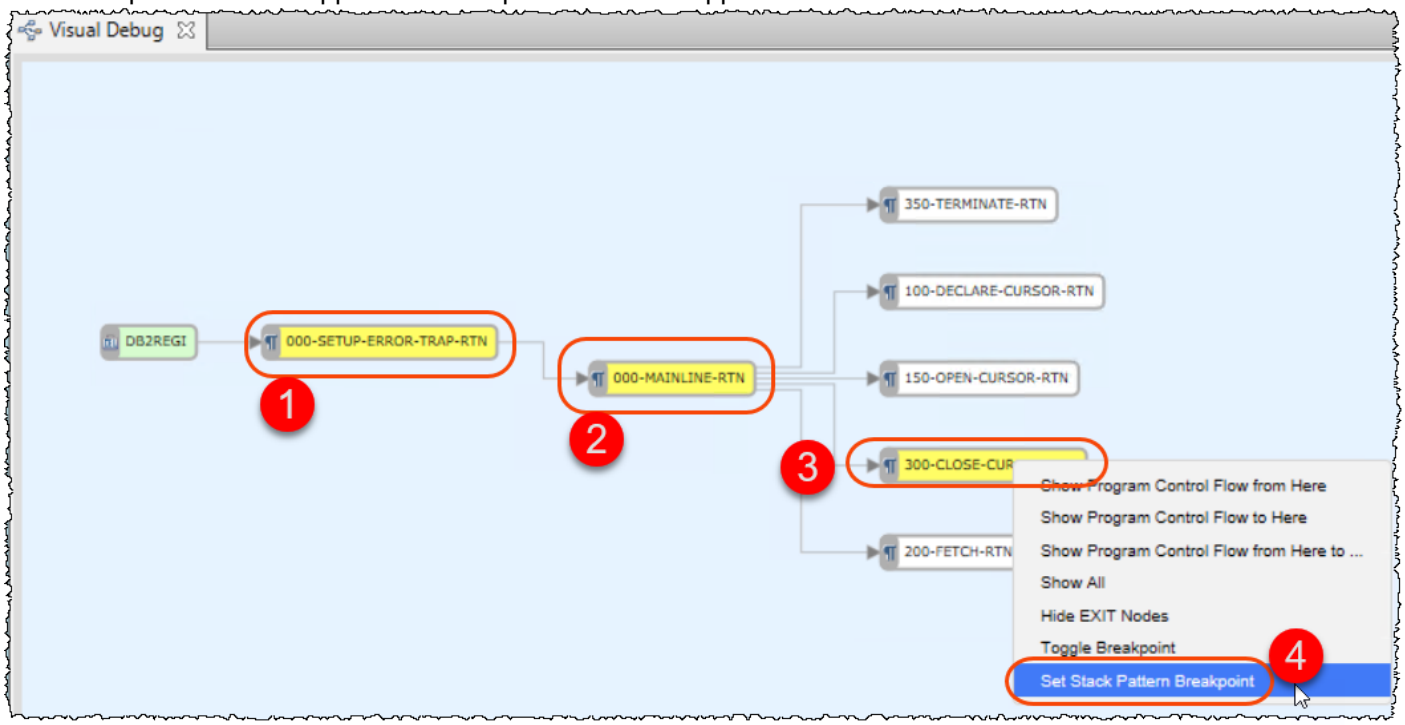
4.2.5 (Optional) Resize the **Visual Debug** view and you will notice that the paragraphs executed are marked as green


Visual Debug
DB2REGI
000-SETUP-ERROR-TRAP-RTN
000-MAINLINE-RTN
350-TERMINATE-RTN
100-DECLARE-CURSOR-RTN
150-OPEN-CURSOR-RTN
300-CLOSE-CURSOR-RTN
200-FETCH-RTN
Those paragraphs were executed (marked as green)
pot01db.jcl
DB2REGI.expanded.cbl
Outline
Variables
Name
no local variables are availa
219 * THE MAINLINE CONTAINS THE DRIVER CODE TO PERFORM OUR DATA
220 * BASE ACCESS AND DISPLAY ROUTINES.
221 PERFORM 100-DECLARE-CURSOR-RTN THRU 100-EXIT.
222 PERFORM 150-OPEN-CURSOR-RTN THRU 150-EXIT.
223 PERFORM 200-FETCH-RTN THRU 200-EXIT
224 UNTIL SQLCODE = +100.

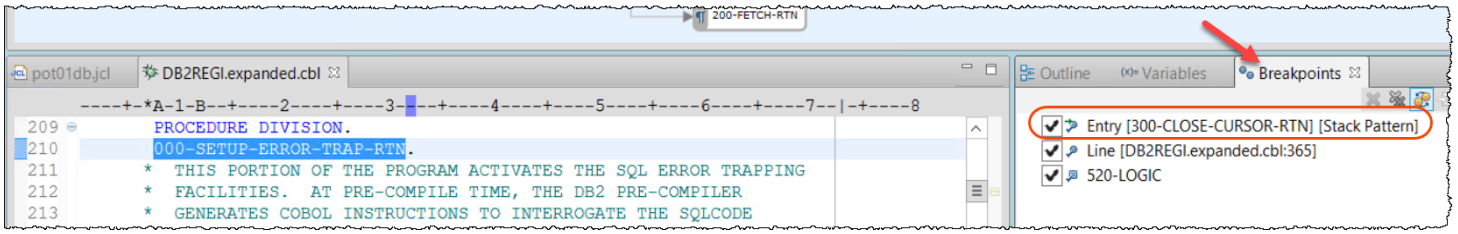
4.2.6 (Optional) Using the *Visual Debug* view, **scroll down**

Press **CTRL** Key and select the 3 paragraphs as shown below, right click and choose **Set Stack Pattern Breakpoint**

This breakpoint that will happen when the path selected happens.

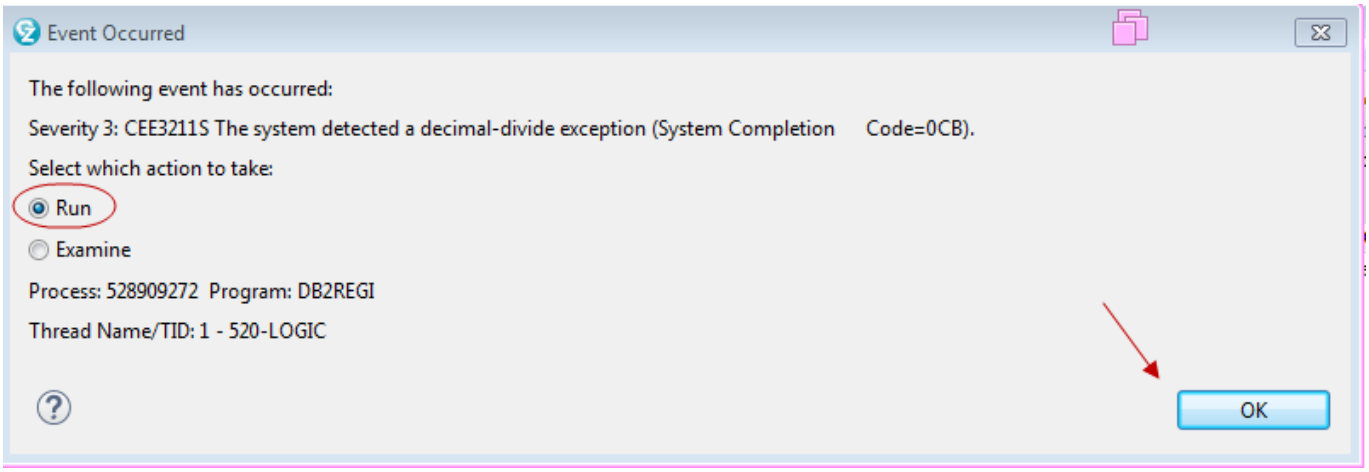


4.2.7 (Optional) Click on **Breakpoints** view and you will see this breakpoint created.



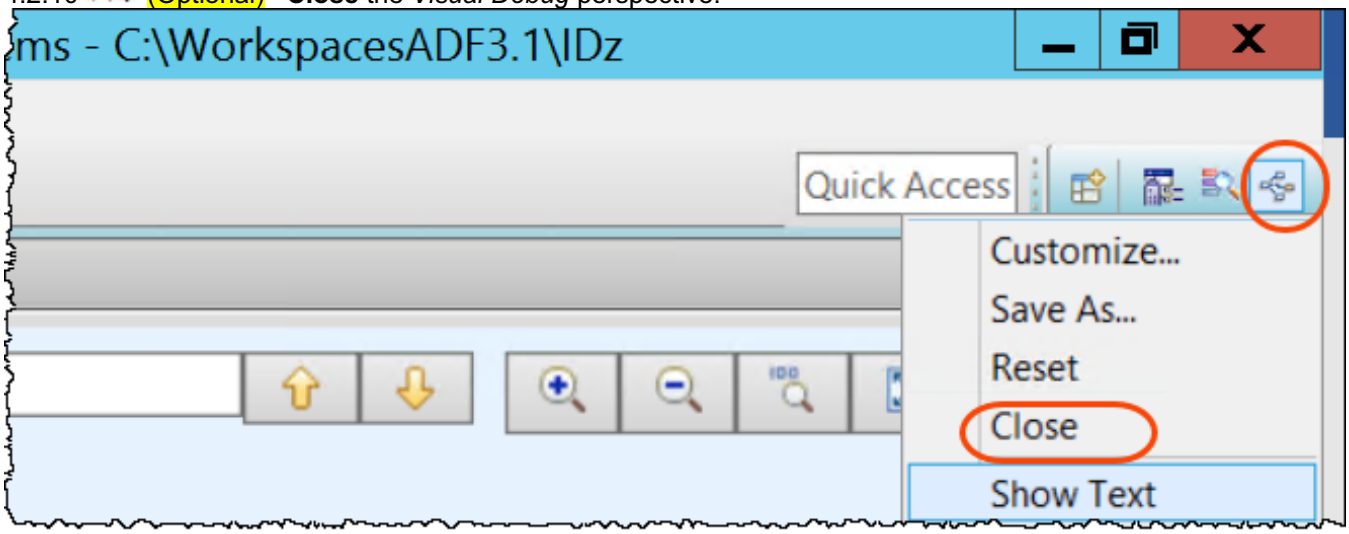
4.2.8 (Optional) Click on (or press **F8**) few times to resume the execution

4.2.9 (Optional) Select **Run** and click **OK** to continue

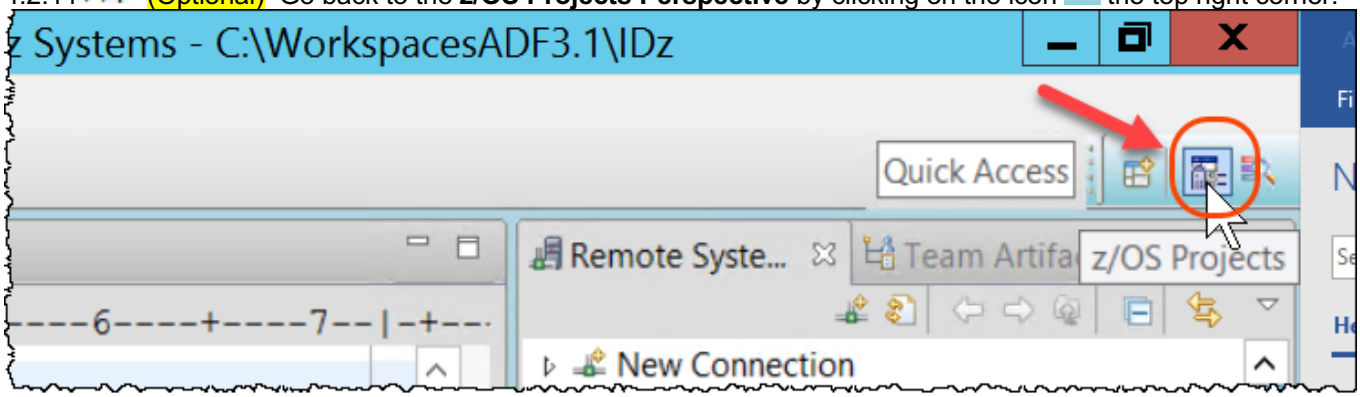


Notice – This breakpoint did not happen since the execution did not execute the 3 selected pargaraphs.

4.2.10 (Optional) **Close** the *Visual Debug* perspective.



4.2.11 (Optional) Go back to the **z/OS Projects Perspective** by clicking on the icon the top right corner.

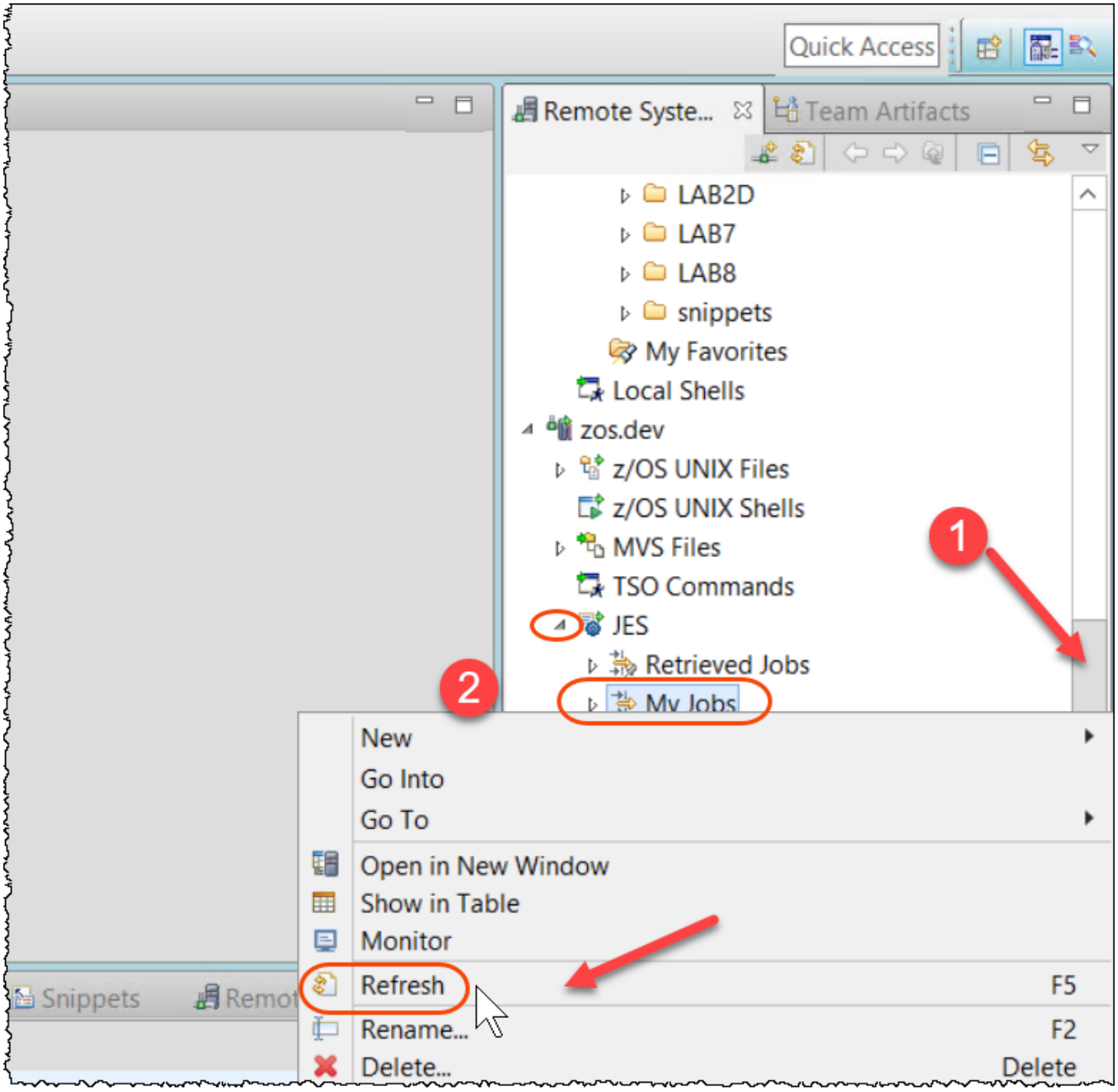


___ 4.3 Accessing the z/OS JES

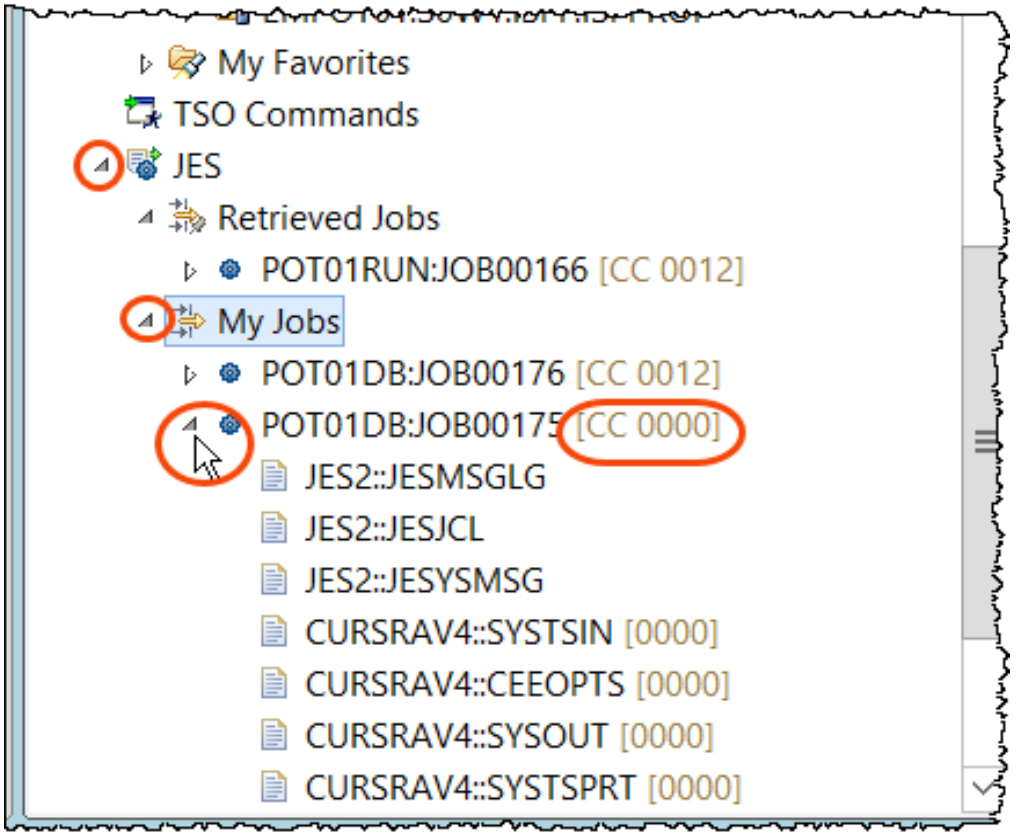
On ISPF many people use *SDSF* for this activity.
You will use the Job Monitor subsystem part of the host components.

4.3.1 Use **Ctrl + Shift + F4** to close all opened editors. Do not save any changes.
Or just click on the . of each opened editor.

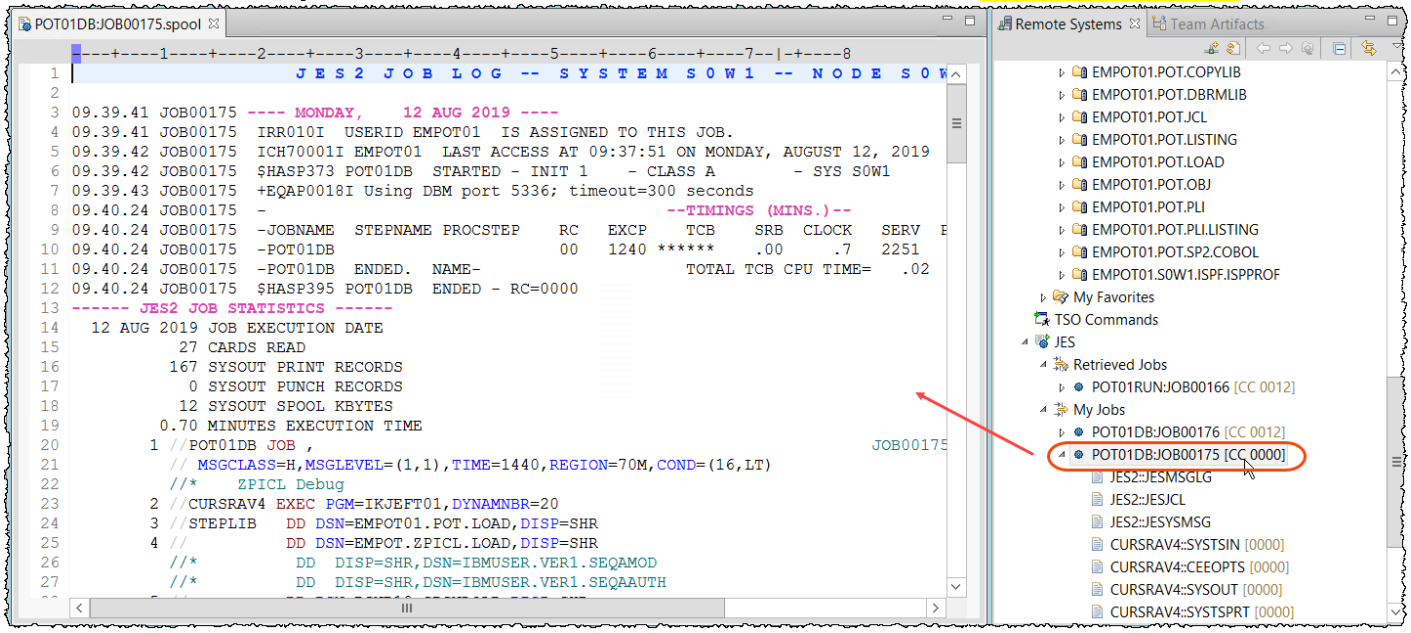
4.3.2 Using the *Remote Systems* view scroll down, expand JES right click on **My Jobs** and select **Refresh**



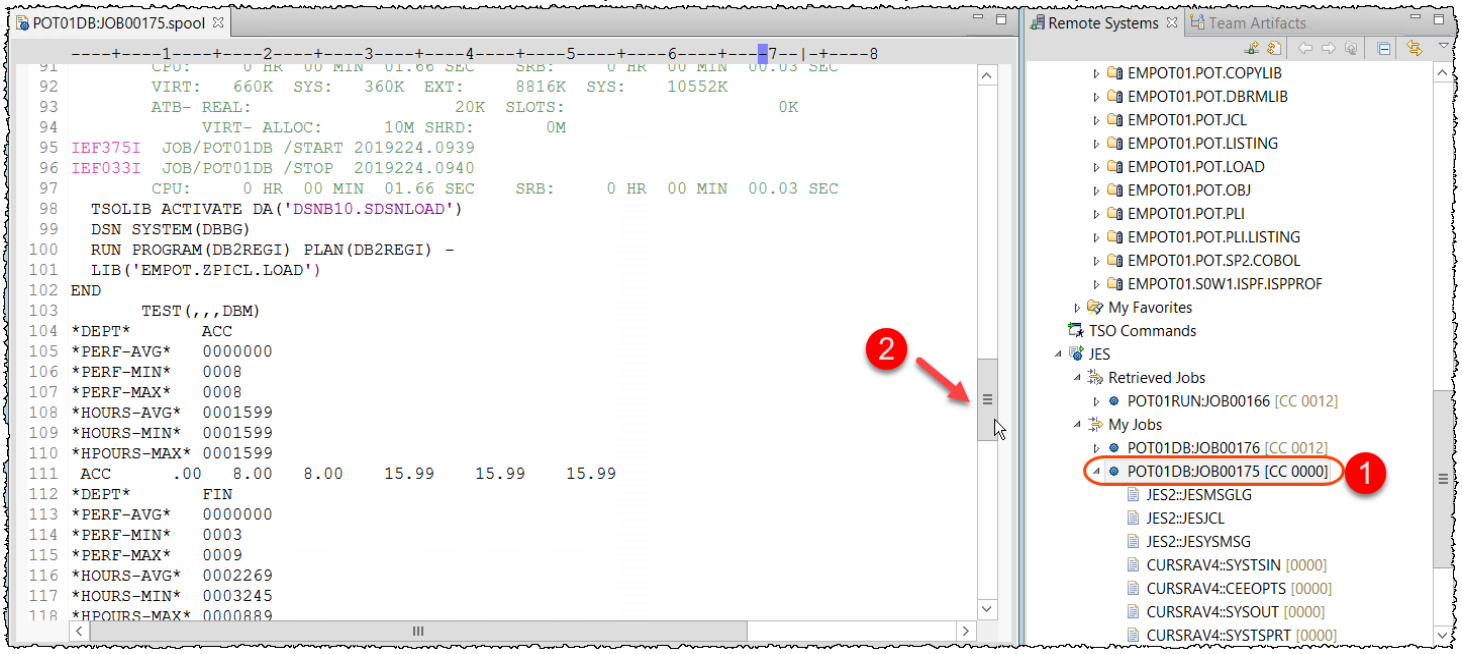
4.3.3 Expand **My Jobs** and the Job **POT01DB:JOB00xxx** that has the CC 0000 (it will be the latest if you did not make the optional step)



4.3.4 On *Remote Systems* view, **double click** on the **POT01DB:JOB00xxx**. that has the CC 0000



4.3.5 **Scroll down** to see the various JOB steps. Notice that a small report has been printed.



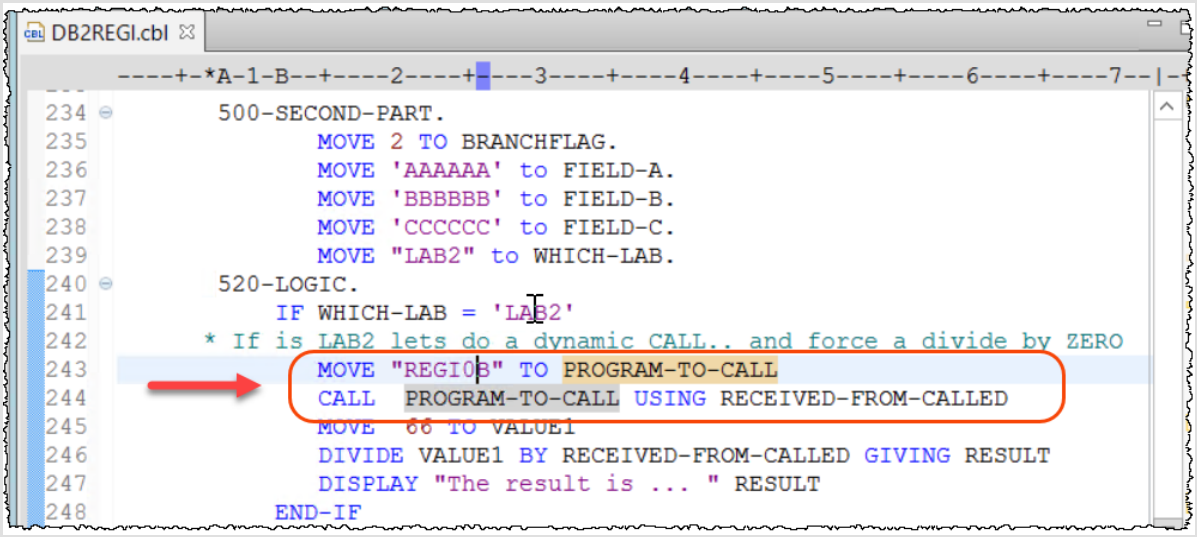
4.3.6 **Close** all opened editors. (**Ctrl + Shift + F4**). Click **No** if asked to save change changes.
Or just click on the . of each opened editor.

Section 5. Modify the COBOL code to fix the bug

You will work with a z/OS member using the editor and now we will be working with the assets located on the z/OS remote system.

What z/OS remote assets you will work with?

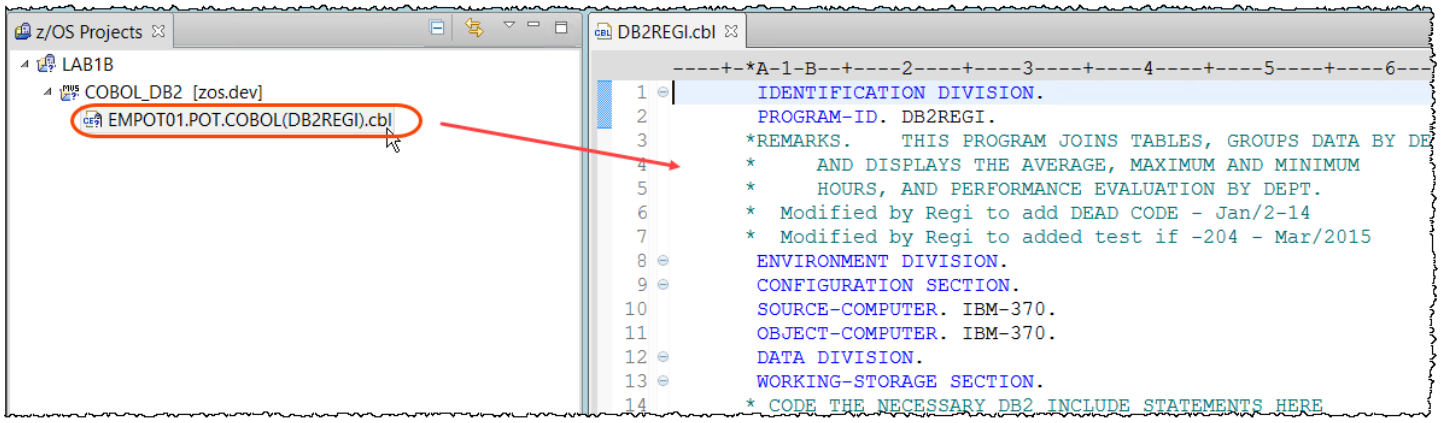
This is a batch program that reads DB2. In addition, this program does a Dynamic call to another COBOL program named **REGI0B**.



___ 5.1 Using the Editor with z/OS COBOL programs

Before fixing the code let's explore some editor capabilities on the main DB2 COBOL program.

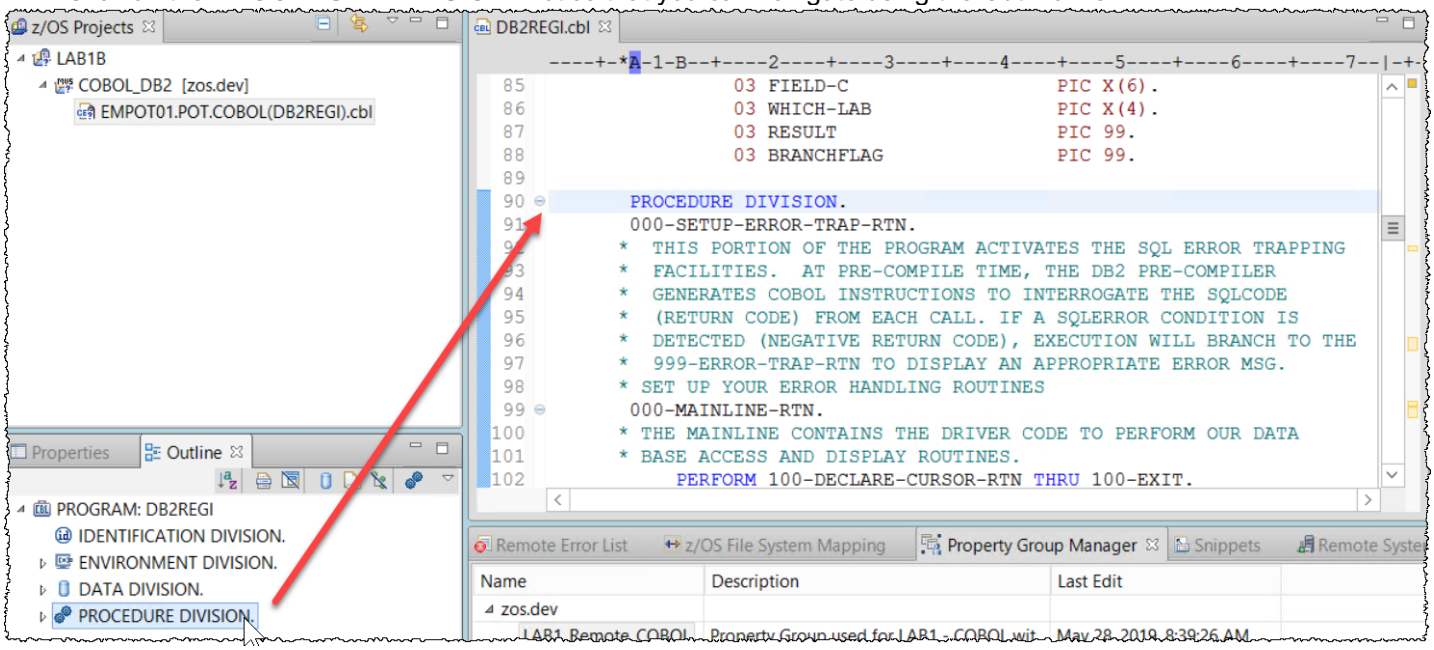
5.1.1 Using the *z/OS Projects* perspective and the *z/OS Projects* view, **double click** on **EMPOT01.POT.COBOL(DB2REGI)** to open the file using the editor.


z/OS Projects
LAB1B
COBOL_DB2 [zos.dev]
EMPOT01.POT.COBOL(DB2REGI).cbl
DB2REGI.cbl
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6---
1 IDENTIFICATION DIVISION.
2 PROGRAM-ID. DB2REGI.
3 *REMARKS. THIS PROGRAM JOINS TABLES, GROUPS DATA BY DE
4 * AND DISPLAYS THE AVERAGE, MAXIMUM AND MINIMUM
5 * HOURS, AND PERFORMANCE EVALUATION BY DEPT.
6 * Modified by Regi to add DEAD CODE - Jan/2-14
7 * Modified by Regi to added test if -204 - Mar/2015
8 ENVIRONMENT DIVISION.
9 CONFIGURATION SECTION.
10 SOURCE-COMPUTER. IBM-370.
11 OBJECT-COMPUTER. IBM-370.
12 DATA DIVISION.
13 WORKING-STORAGE SECTION.
14 * CODE THE NECESSARY DB2 INCLUDE STATEMENTS HERE

5.1.2 Click on the **Outline** tab (bottom and left) to see the *Outline* view.

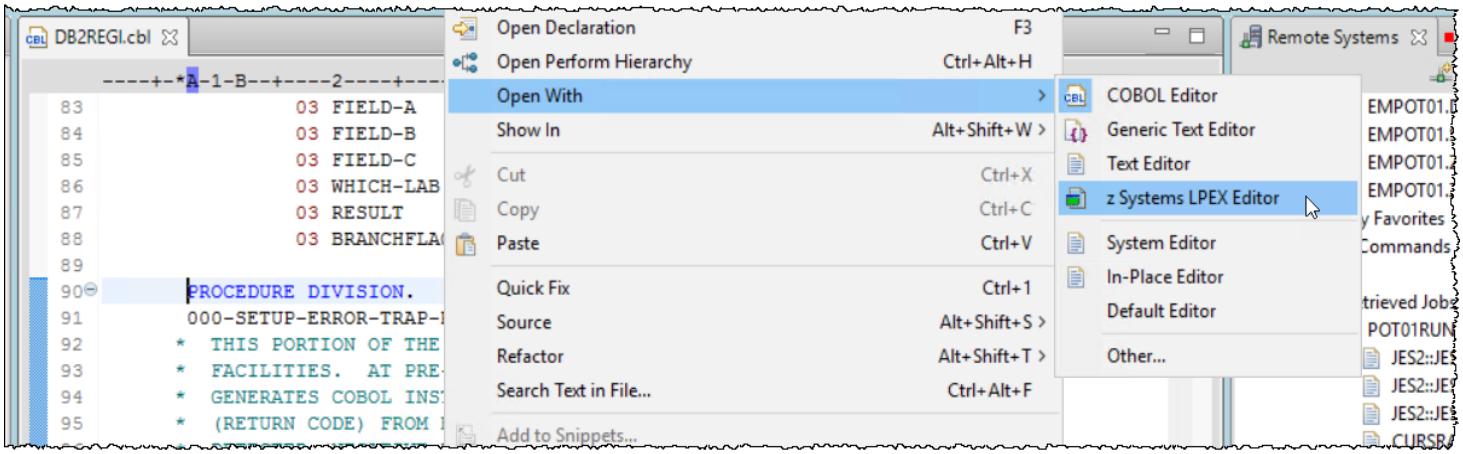
Using the editor, browse the program and notice that the contents of the *Outline view* are synchronized with the COBOL source code and vice versa.

Click on the **PROCEDURE DIVISION.** Notice that you can navigate using the *Outline* view.

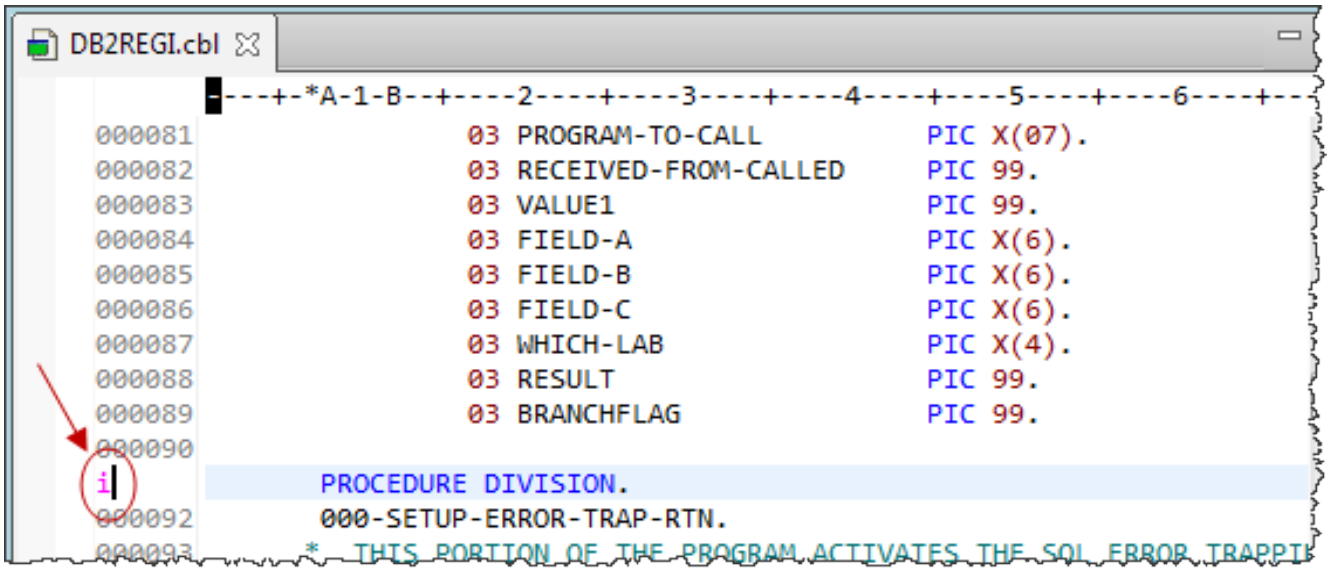


5.1.3 If you prefer the ISPF editor, you might use another IDz editor called “LPEX editor”.

To switch the editor, right click on the program and select **Open with > z Systems LPEX Editor**



5.1.4 Note that the column on left are similar as the prefix area of the ISPF editor..
You may type commands to add lines etc..
Like **I** to insert a line, **d** to delete, **m** to move, etc. **Try it**.


DB2REGI.cbl
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+--
000081 03 PROGRAM-TO-CALL PIC X(07).
000082 03 RECEIVED-FROM-CALLED PIC 99.
000083 03 VALUE1 PIC 99.
000084 03 FIELD-A PIC X(6).
000085 03 FIELD-B PIC X(6).
000086 03 FIELD-C PIC X(6).
000087 03 WHICH-LAB PIC X(4).
000088 03 RESULT PIC 99.
000089 03 BRANCHFLAG PIC 99.
000090
i PROCEDURE DIVISION.
000092 000-SETUP-ERROR-TRAP-RTN.
000093 * THIS PORTION OF THE PROGRAM ACTIVATES THE SQL ERROR TRAPPI

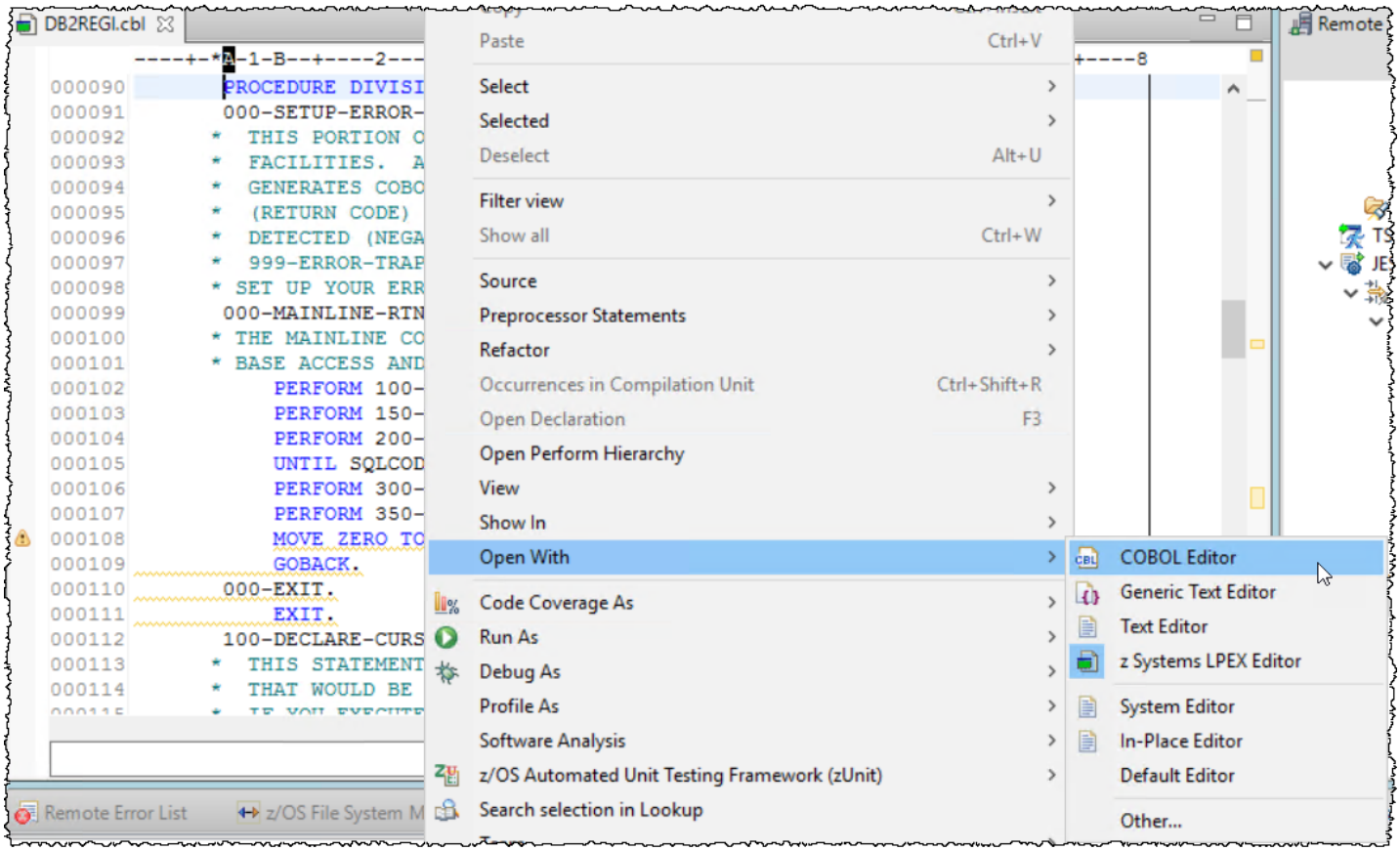
5.1.5 Switch back to **COBOL editor** (since the screen captures uses the COBOL editor)

Right click on the program and select **Open with > COBOL Editor**

Click **No** for saving changes**.**

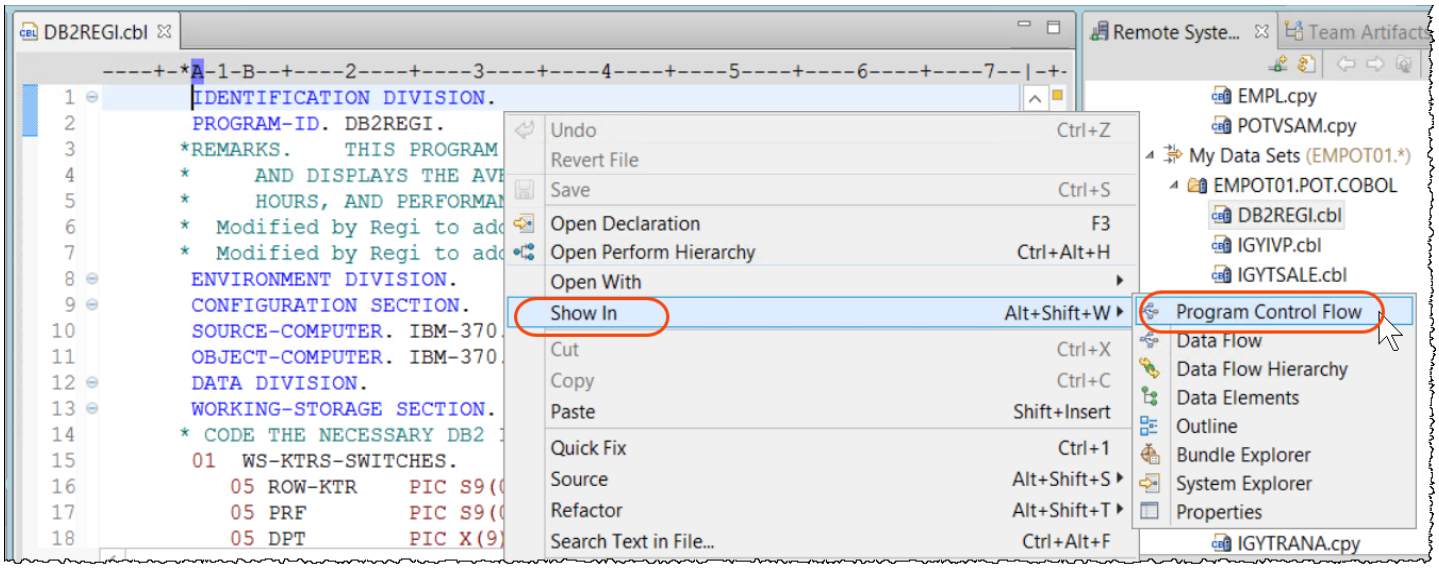
This editor is more like the Java Editor and gives you more capabilities. When you see all advantages, probably you will prefer this editor than the ISPF like editor (LPEX)..

Please keep this program opened as we will work on it later.



5.1.6 Let's see the program in more details..

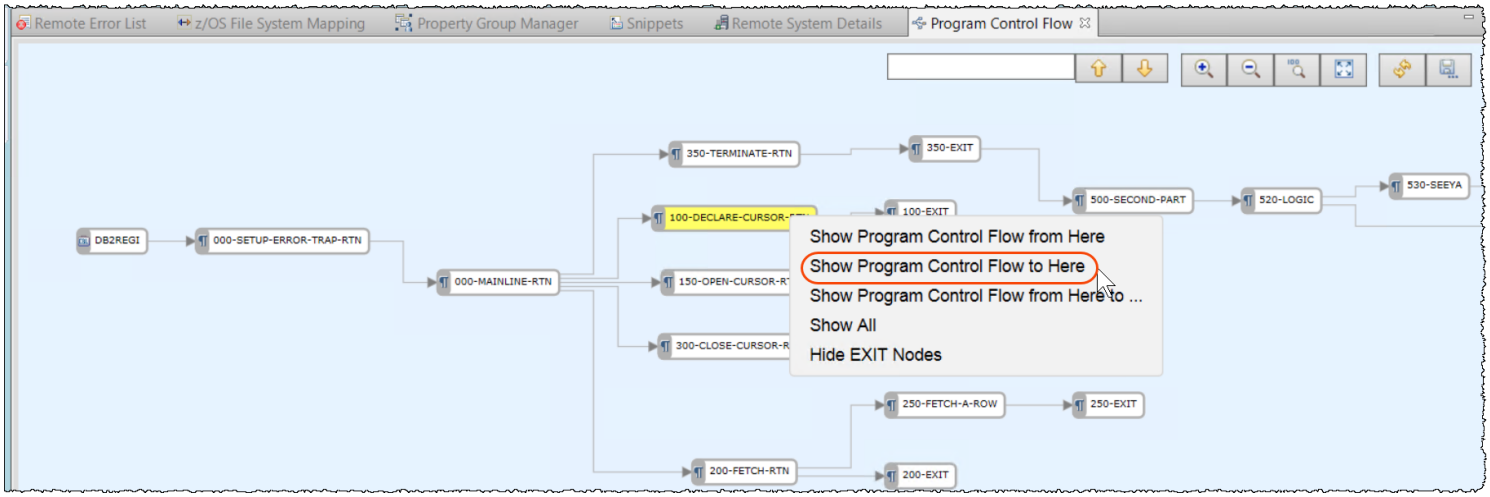
Right click on the editor and select **Show In → Program Control Flow**


DB2REGI.cbl
Remote Syste...
Team Artifacts
EMPL.cpy
POTVSAM.cpy
My Data Sets (EMPOT01.*)
EMPOT01.POT.COBOL
DB2REGI.cbl
IGYIVP.cbl
IGYTSALE.cbl
IGYTRANA.cpy
Undo Ctrl+Z
Revert File
Save Ctrl+S
Open Declaration F3
Open Perform Hierarchy Ctrl+Alt+H
Open With
Show In Alt+Shift+W
Cut Ctrl+X
Copy Ctrl+C
Paste Shift+Insert
Quick Fix Ctrl+1
Source Alt+Shift+S
Refactor Alt+Shift+T
Search Text in File... Ctrl+Alt+F
Program Control Flow
Data Flow
Data Flow Hierarchy
Data Elements
Outline
Bundle Explorer
System Explorer
Properties

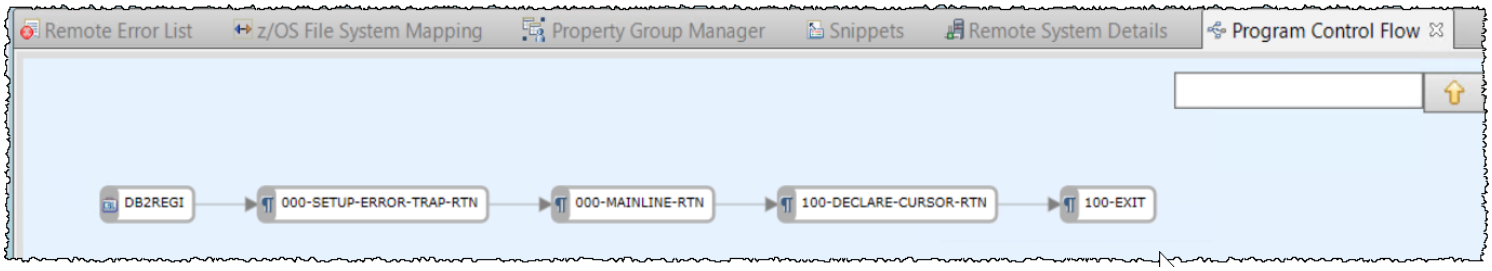
5.1.7 See the results under *Program Control Flow* view.

⏭ Double click "**Program Control Flow"** title to have a bigger diagram (full page).

⏭ Right-click on **100-DECLARE-CURSOR-RTN** box and select **Show Program Flow Control to here.**



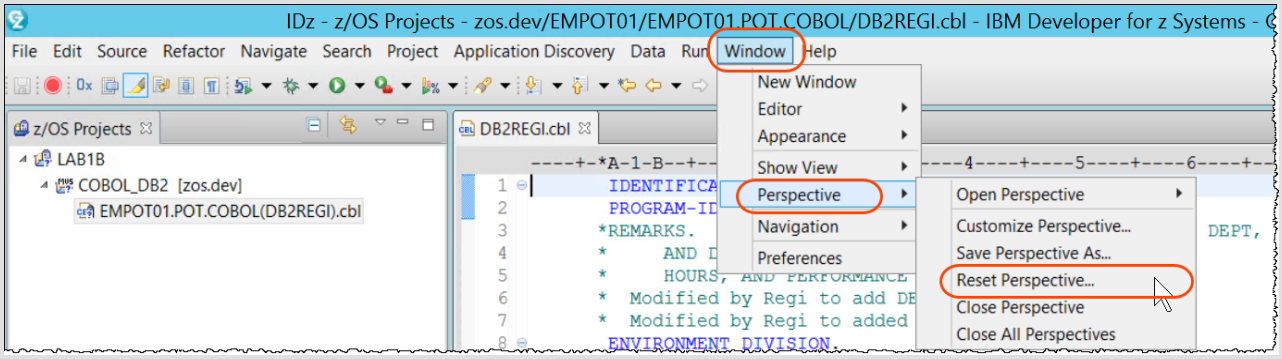
5.1.8 It shows the path to reach this paragraph.. Nice uh? That helps with extremely complex diagrams



The perspectives are not the same as here? Got confused with the opened views?

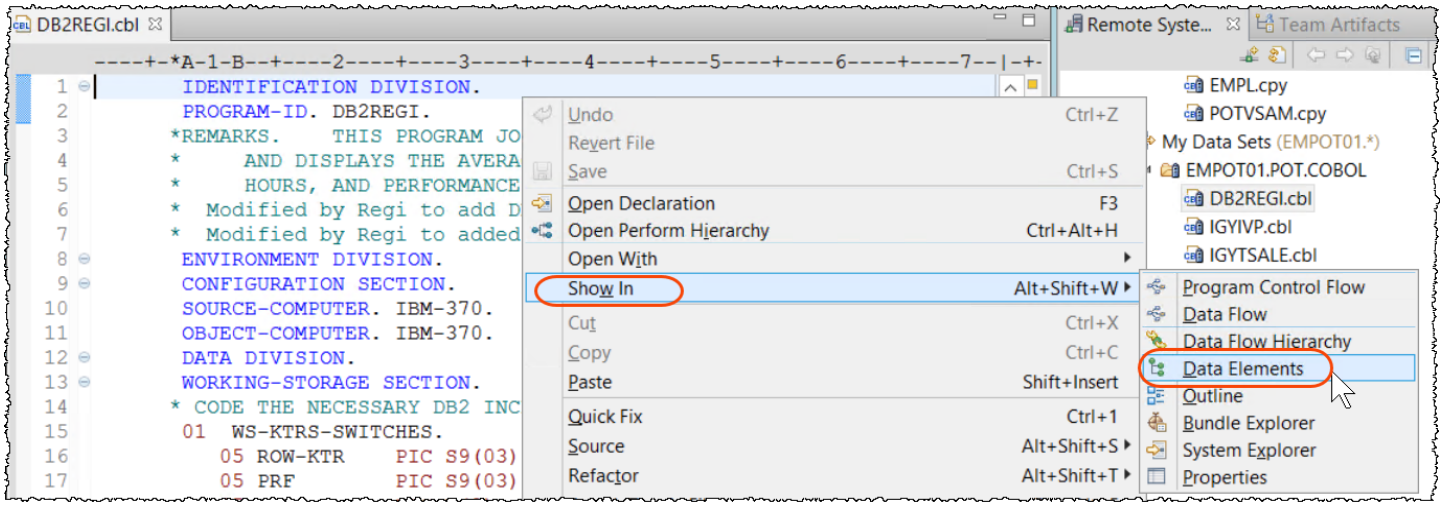
At any time, you may restore the perspective to the default. You may need that when you did mistakes and closed views you should not.

Just select **Windows > Perspective > Reset Perspective…** as below:

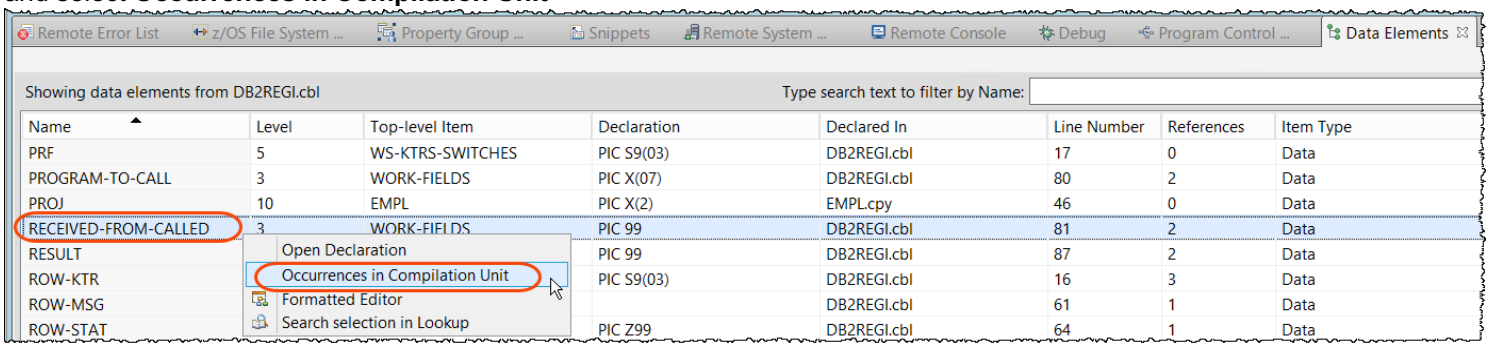


5.1.9 ⏭ Double click "**Program Control Flow"** title to have the diagram smaller.

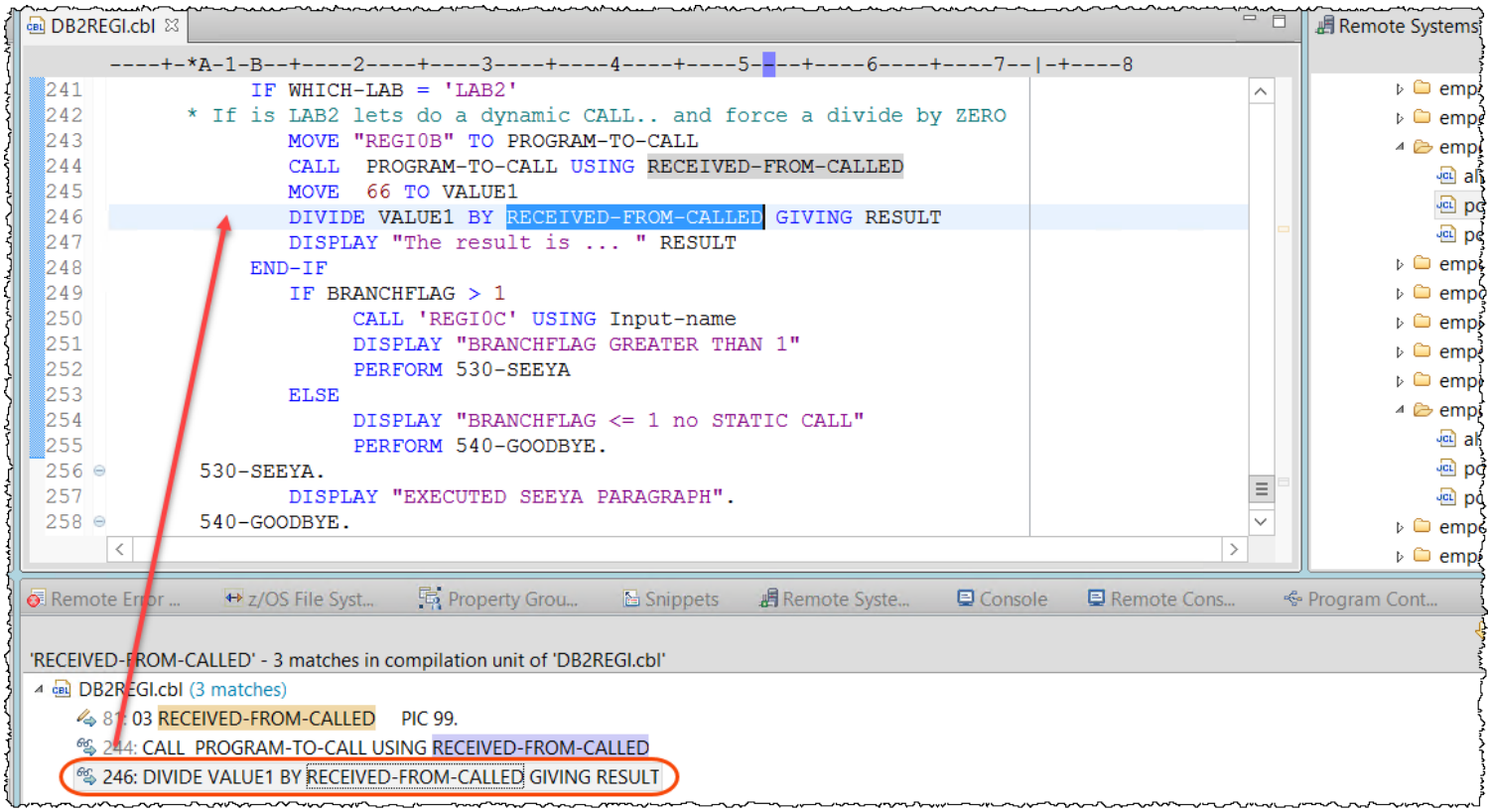
⏭ Right click again on the editor and select
Show In → Data Elements



5.1.10 ⏭ On *Data Elements* view, resize the view, scroll down and right click on **RECEIVED-FROM-CALLED** and select **Occurrences in Compilation Unit**



5.1.11 ⏭ Double click on line **246**. It will show in the line that is abending.
Notice the colors. When the variable is referenced, it is blue and when it is modified is brown.

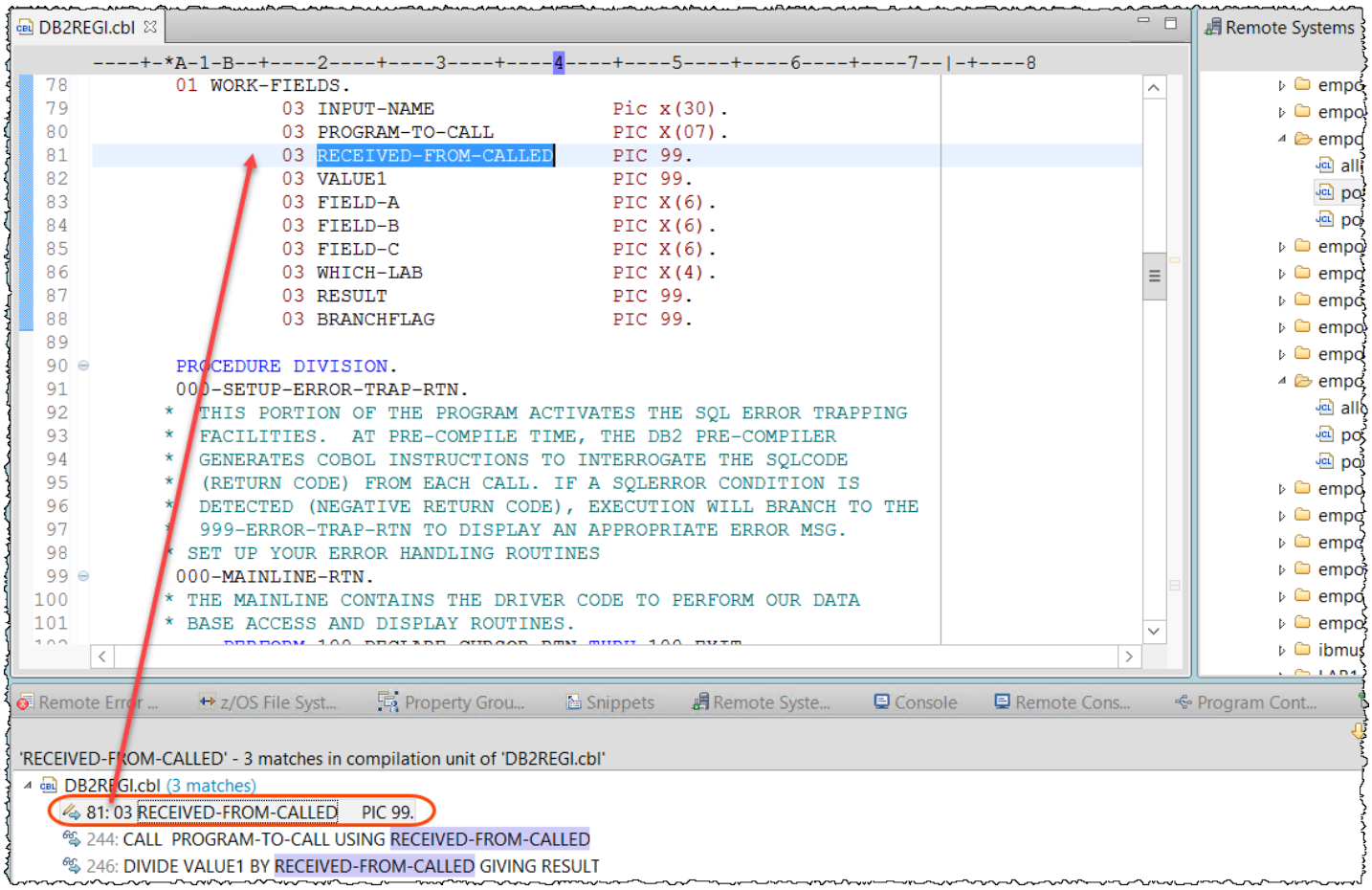


You need to fix the called program named **REGI0B** that is returning 0 on this variable as seen before with Fault Analyzer.

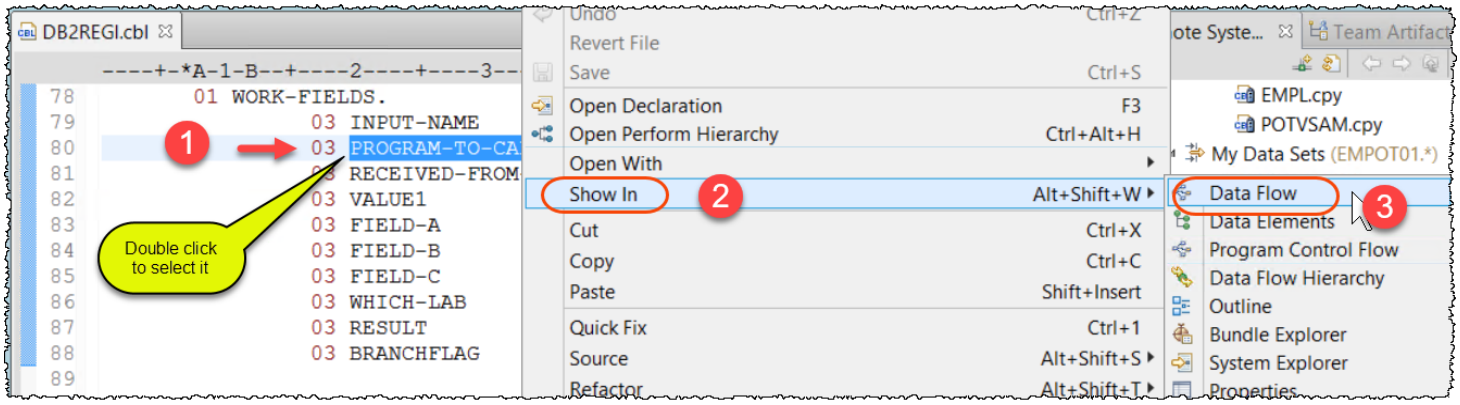
___ Exploring Data Flow

This capability is cool. *Program data flow* provides a graphical and hierarchical view of the data flow within a COBOL program. You can use this feature to examine how a data element is populated, modified, or written elsewhere.

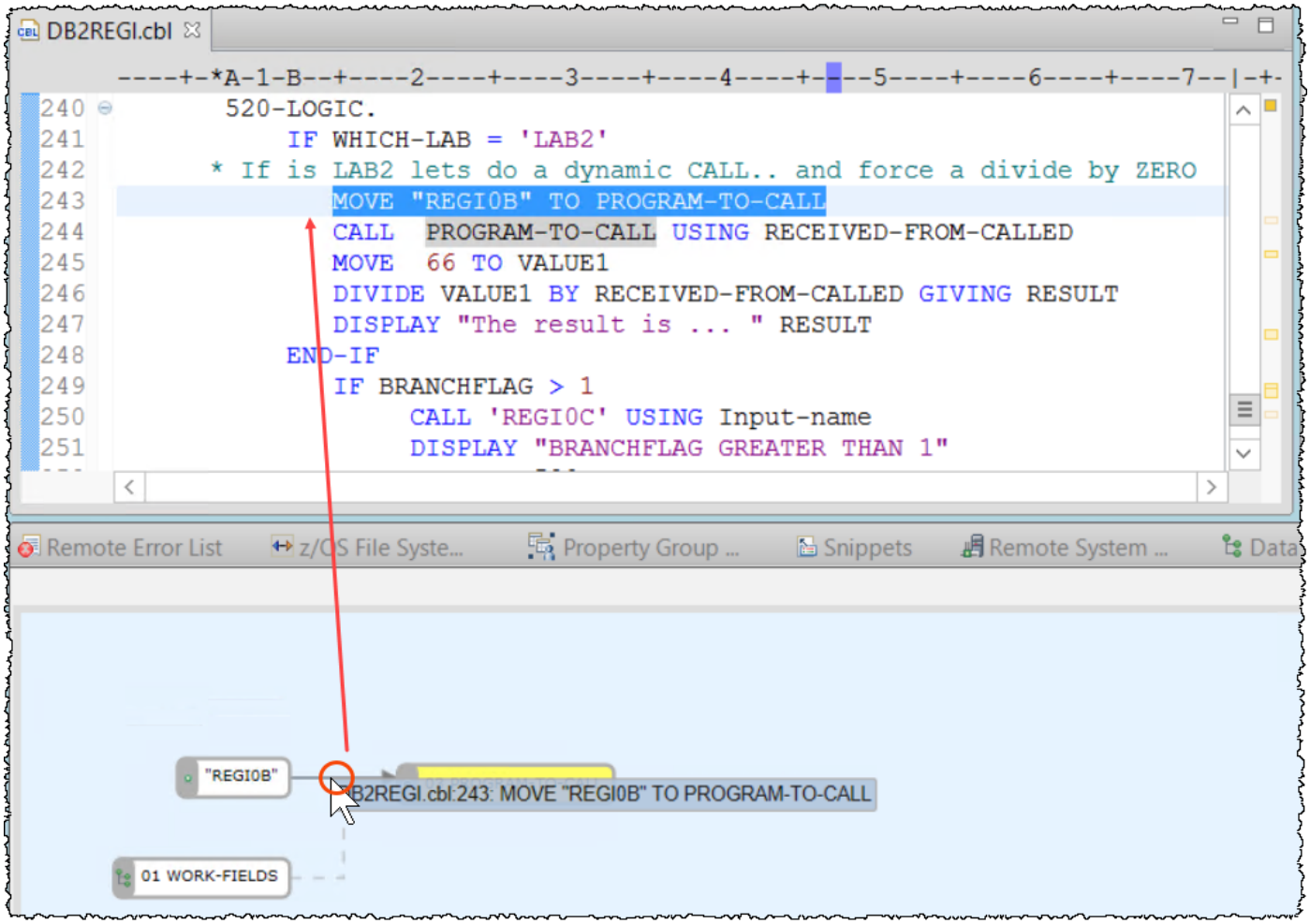
5.1.12 Double click on line **81**. It will show where *RECEIVED-FROM-CALLED* is defined.



5.1.13 Select **PROGRAM-TO-CALL (on line 80)** double clicking on it, **right click** and select **Show In > Data Flow**. (the options order can differ from the screen capture here).



5.1.14 ⏭ Move the cursor to the line between **REGI0B** and **03 PROGRAM-TO-CALL**.
⏭ Clicking in this line shows the program statement where "*REGI0B*" is moved to *PROGRAM-TO-CALL*.



5.1.15 ⏭ **Close** all opened editors if still opened. (**CTRL + Shift + F4**)
Or just click on the ⊠ of each opened editor.

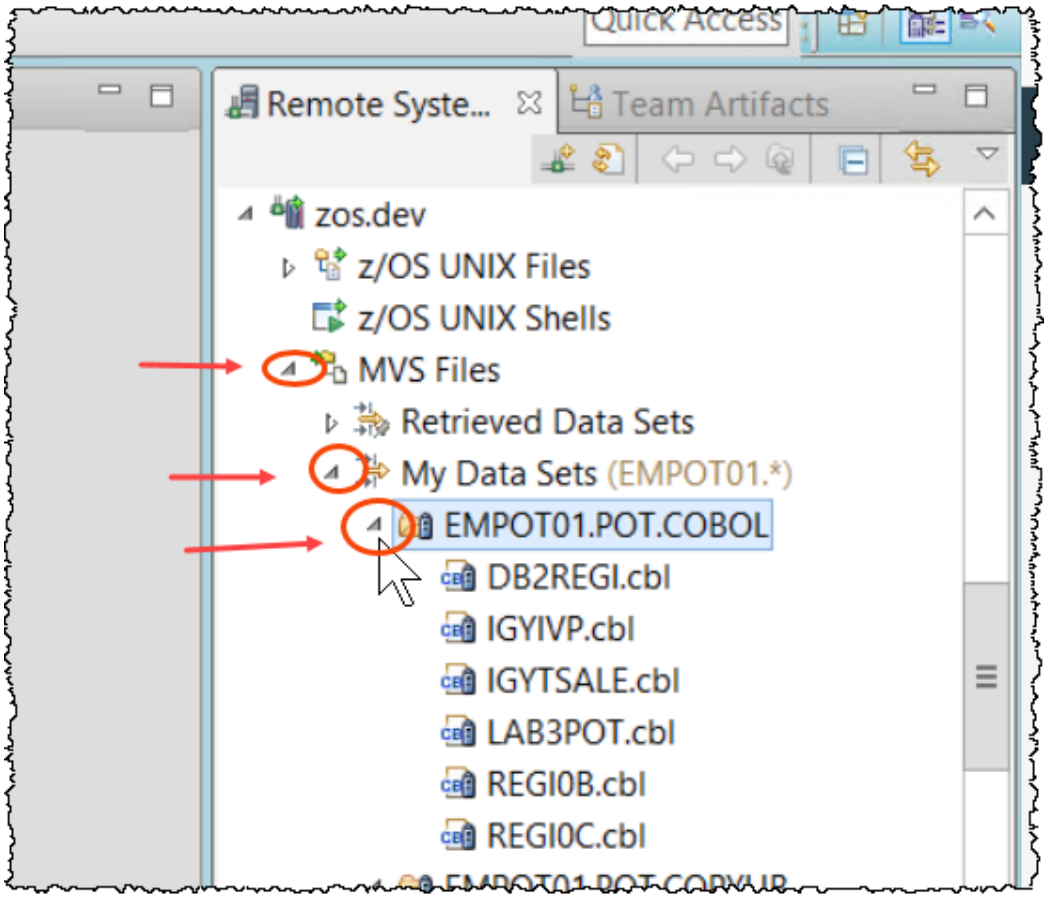
___ 5.2 Fixing the program REGI0B

You need to fix the called program named REGI0B that is returning 0 on this variable as seen before with Fault Analyzer and the debugger.

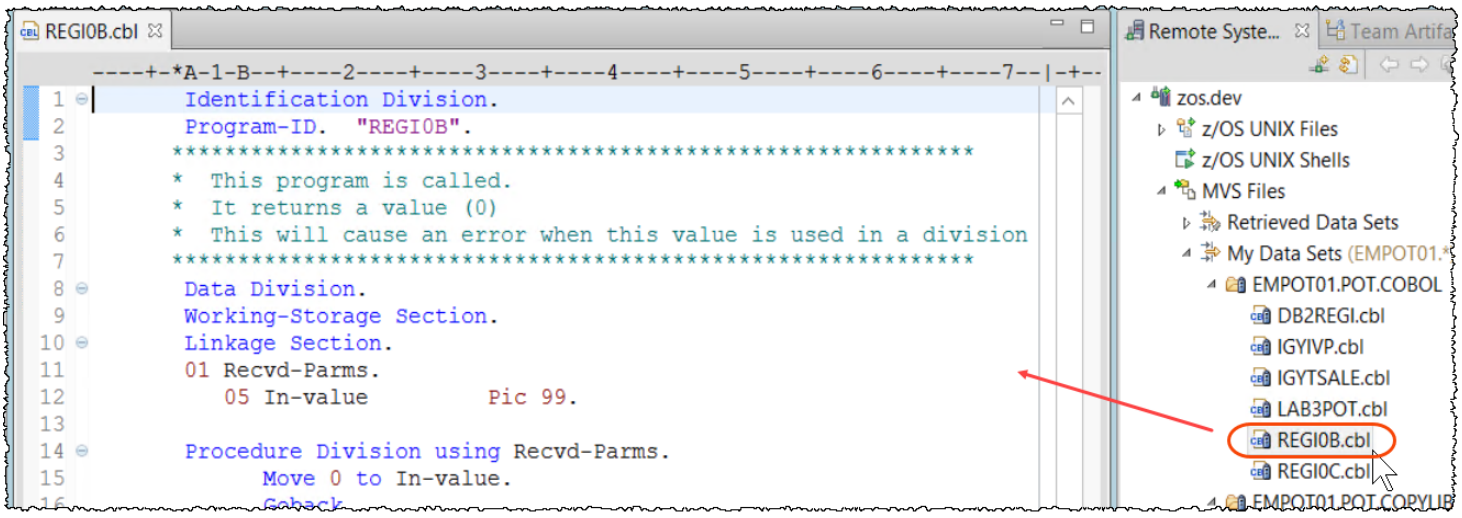
This program is on your ***PDS EMPOT01.POT.COBOL***. .

5.2.1 ⏭ Using *Remote Systems* view **scroll back**, expand **MVS Files, My Data Sets** and

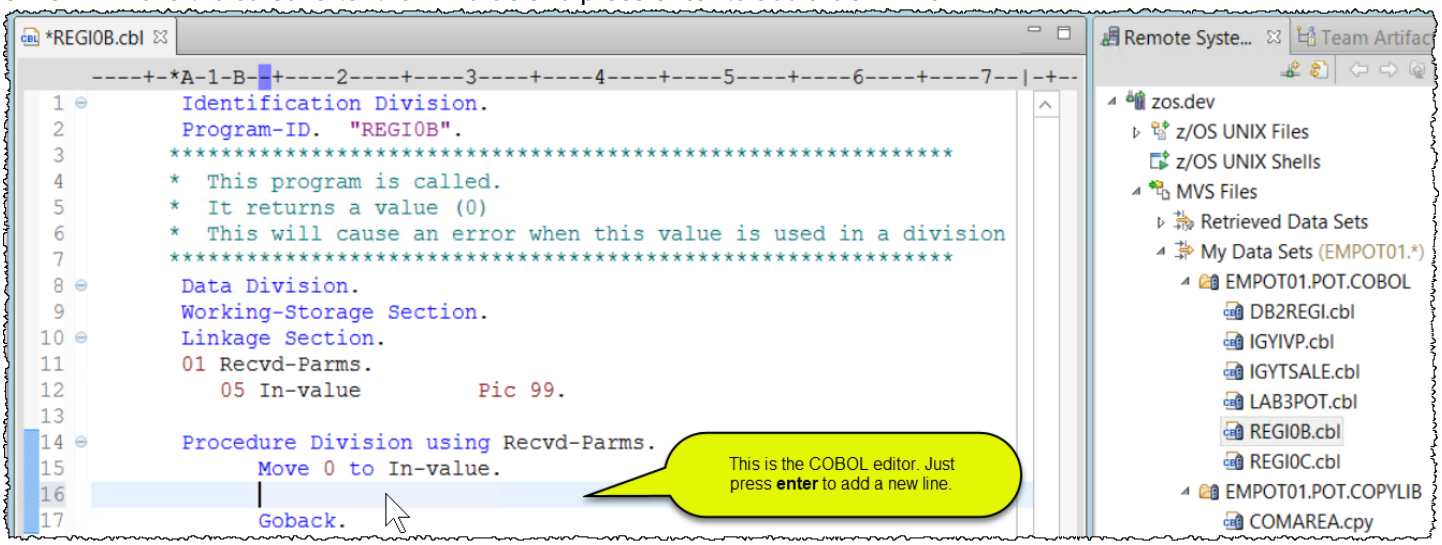
EMPOT01.POT.COBOL. if not already expanded.



5.2.4 **Double** click on the program **`REGI0B.cbl`** to edit it.



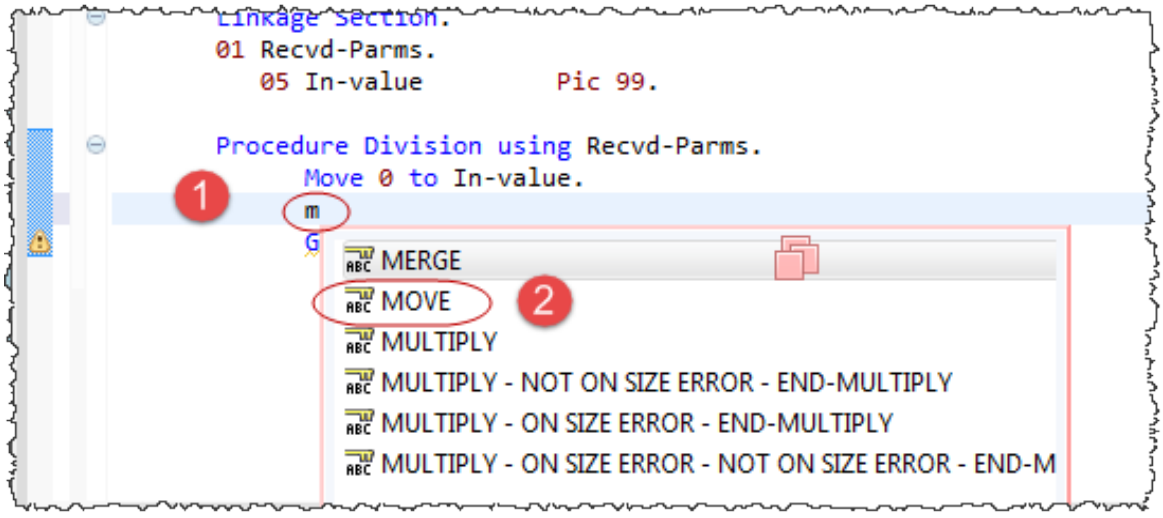
5.2.5 Move the cursor after the **In-value** and press **enter** to add a blank line



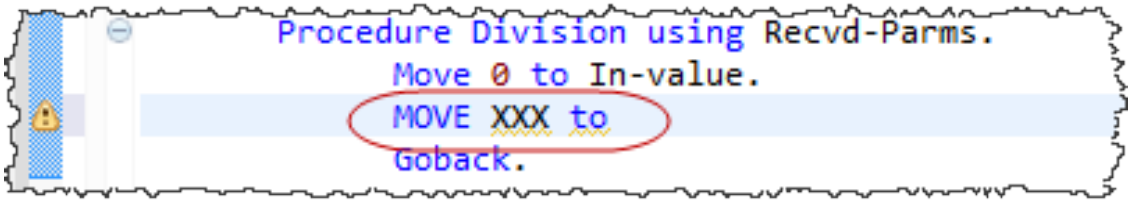
5.2.6 You can now take advantage of the *editor content assist...*

On the column 12 as shown below, type **m** and press **Ctrl + Space**

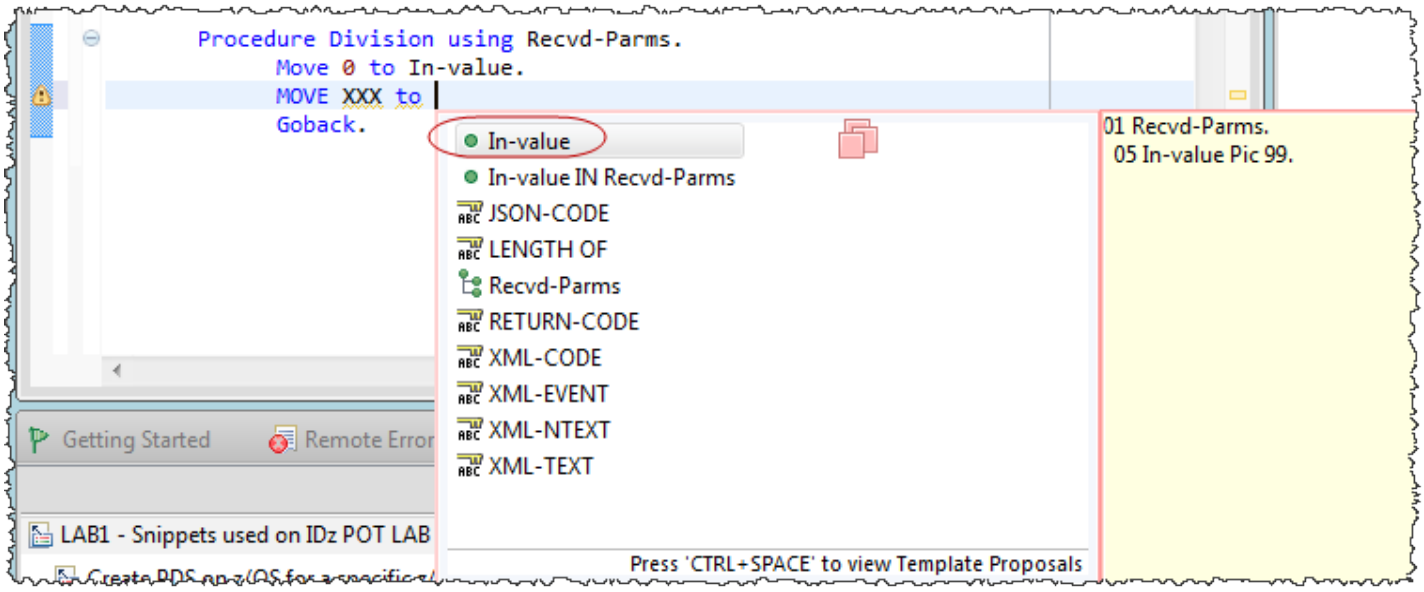
5.2.7 Select the statement **MOVE** (you can use the mouse double click or select and press enter)



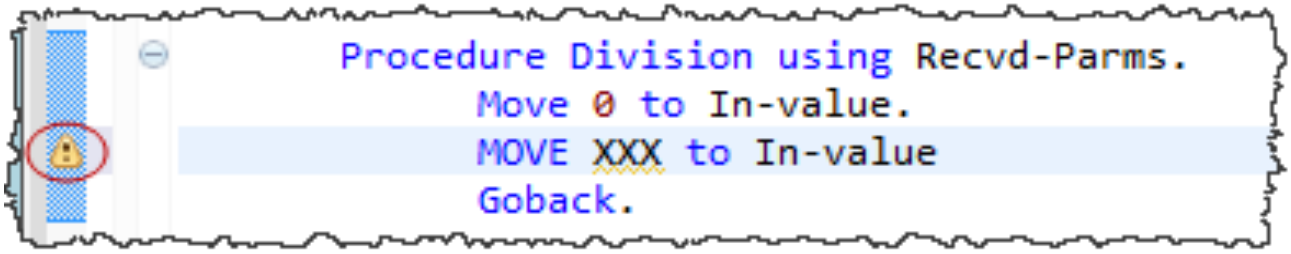
5.2.8 Type "**XXX to**"



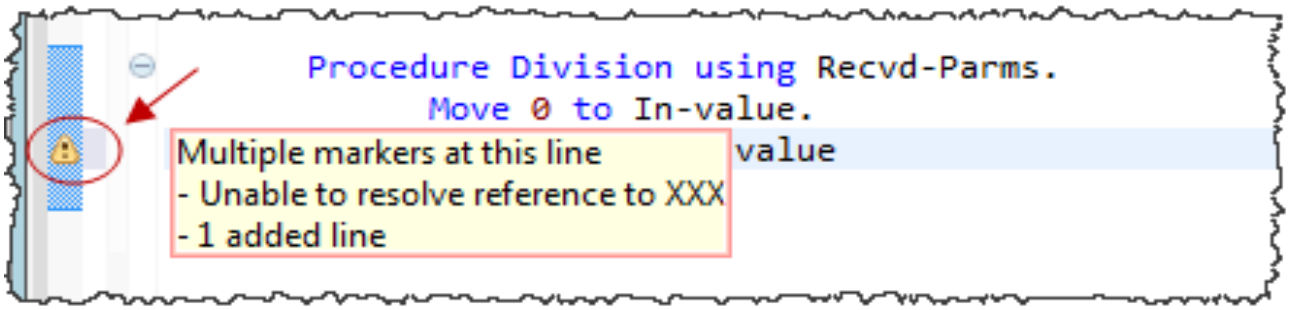
5.2.9 Press **Ctrl + Space** to list the variables and select **In-value**.
Notice that variables could be defined in copybooks.



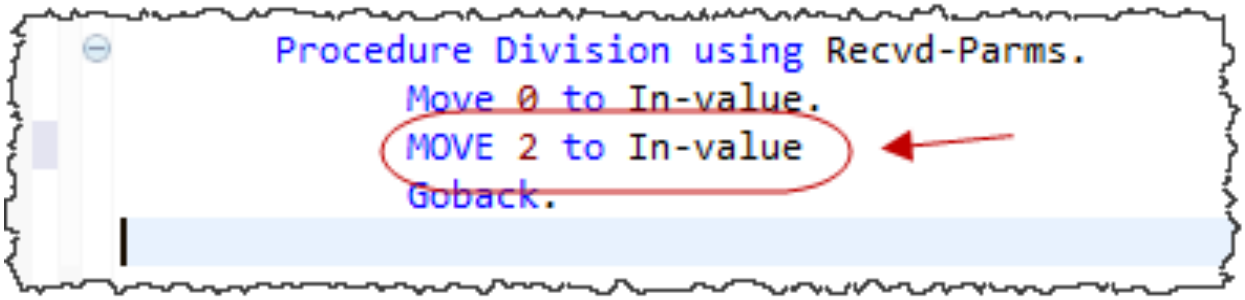
A yellow mark shows that something is wrong.



5.2.10 Move the cursor to the yellow mark to see what the error is.
You are finding this error without using the z/OS compilation. If you were using ISPF you would see that only after submitting this JOB to the z/OS COBOL compiler.
Productivity and z/OS CPU saving. You are using the power of the smart editor.



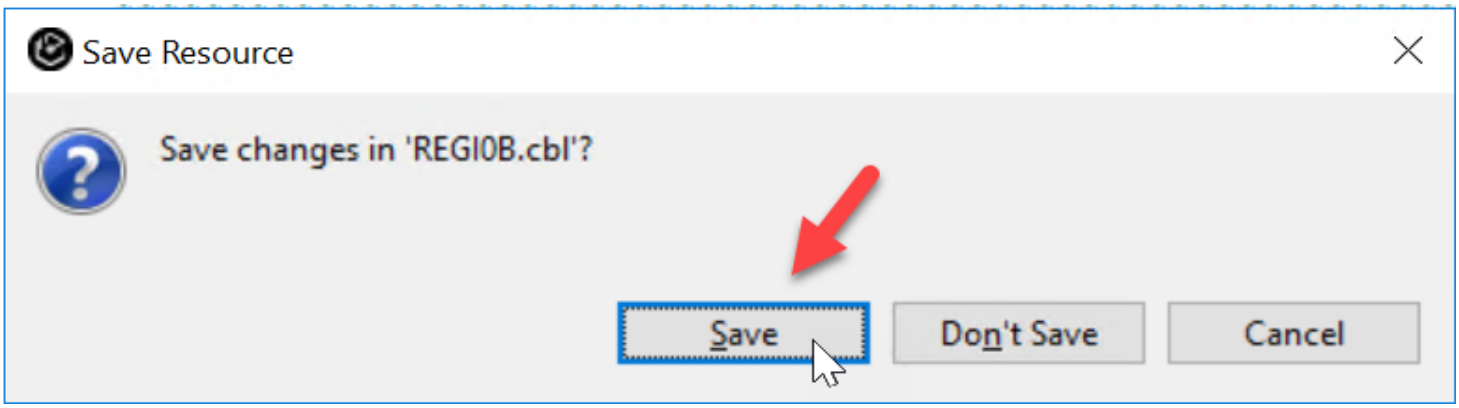
5.2.11 ⏭ Change from **XXX** to **2** as below. Note that the icon will go away.



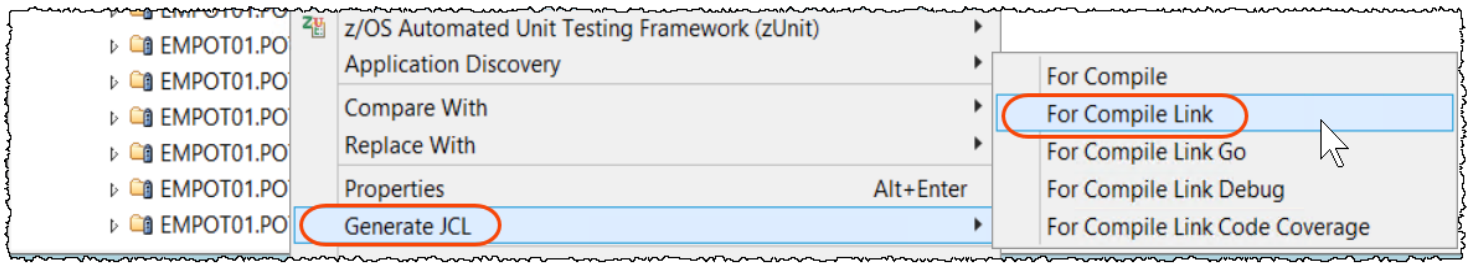
___ 5.3 Compile and link REGI0B.

You need to compile and link the program that you just changed. You could use a JCL or let IDz generate a JCL for you. Once you have a Property Group associated to the COBOL code a JCL file could be generated on the fly when you need it. You have associated the property Group at step 7.2.1 above.

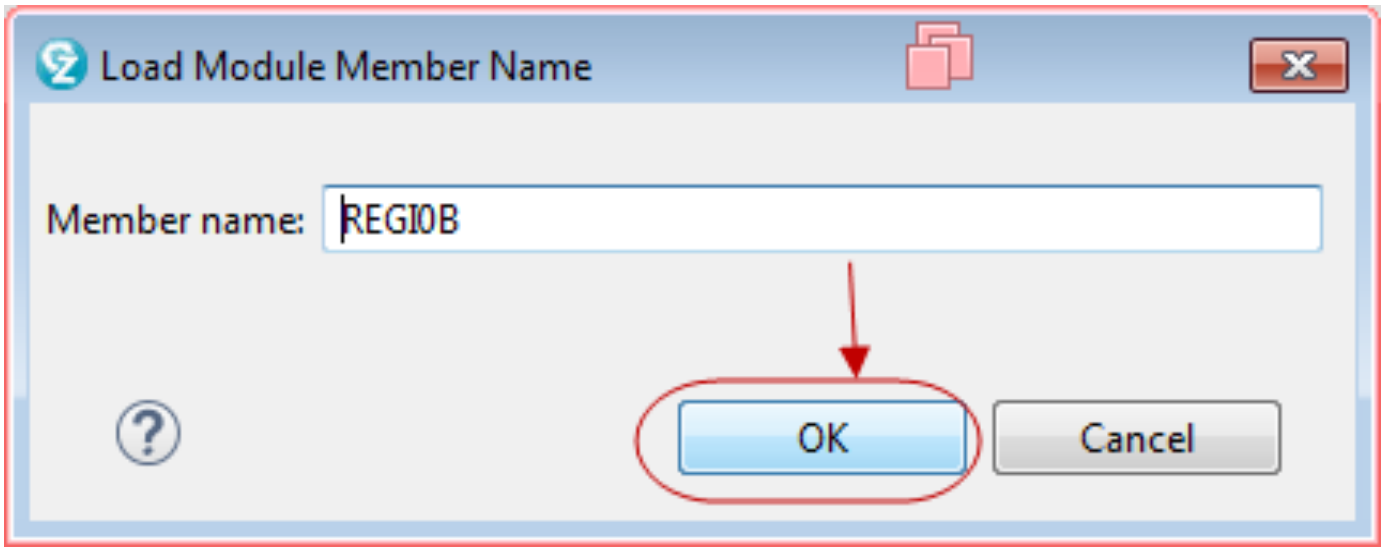
5.3.1 ⏭ Click **Ctrl + Shift + F4** to close the editors. Or just left click on the . of each opened editor.
⏭ Click **Save** to save the changes.



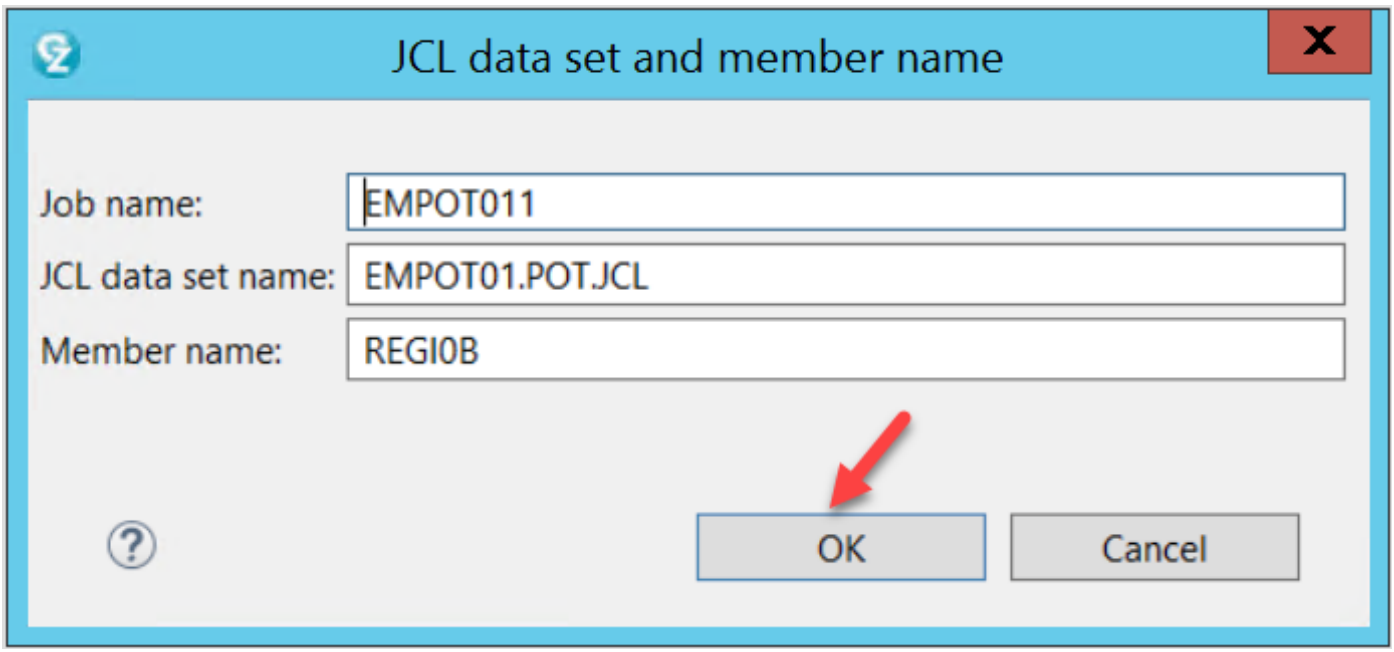
5.3.2 Using the Remote Systems view **right click** on **REGI0B.cbl** and select **Generate JCL > For Compile Link** .



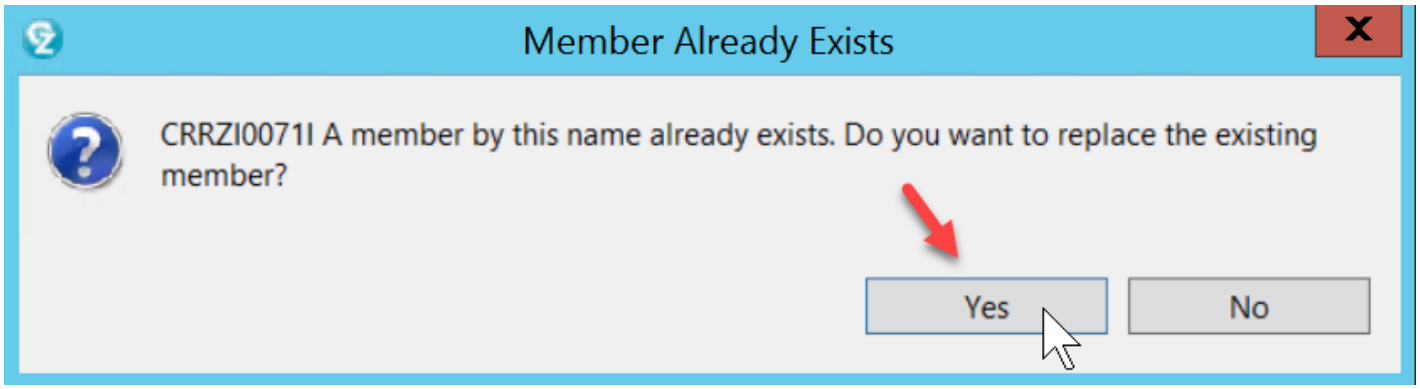
5.3.3 Accept the default member name and click **OK**



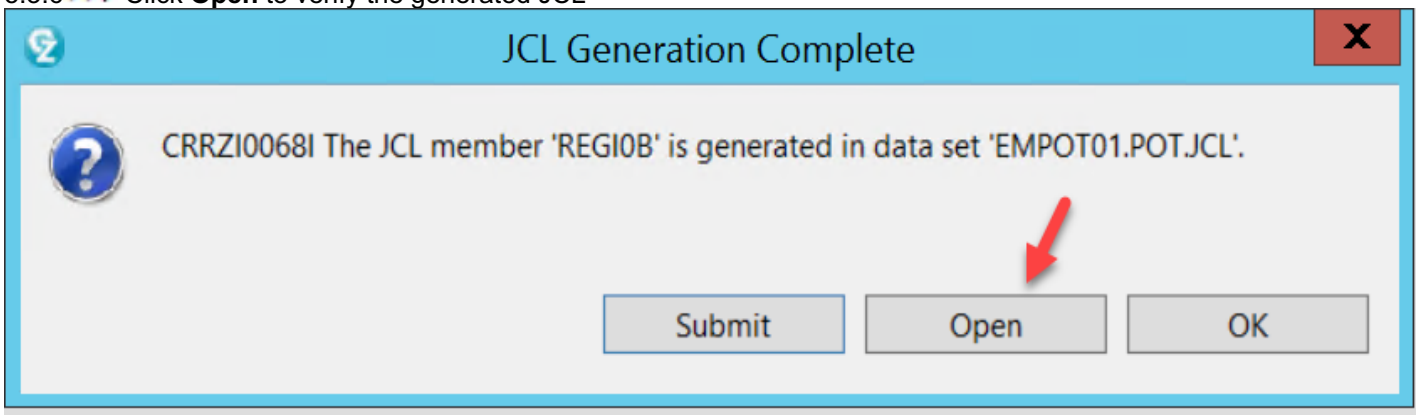
5.3.4 Accept the default and click **OK**



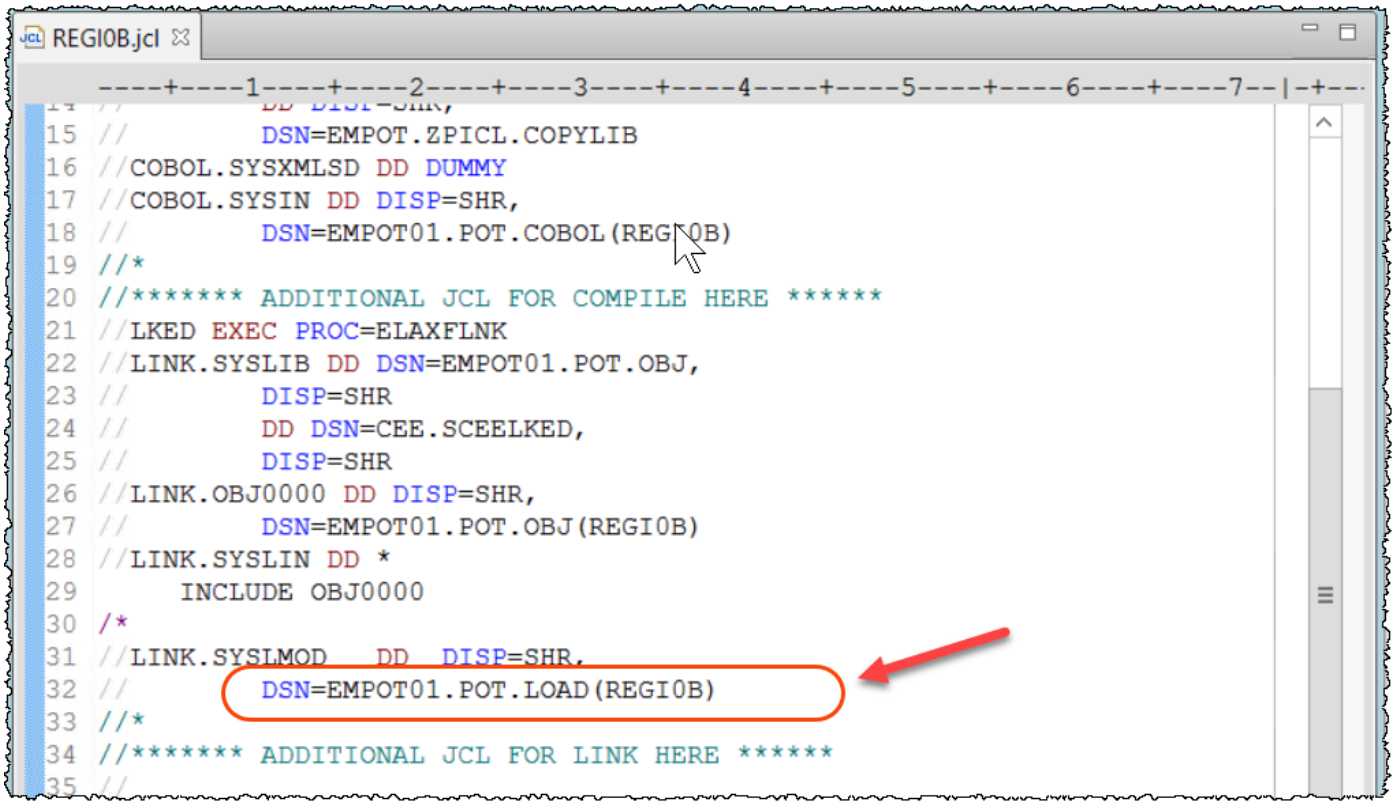
5.3.5 Click **Yes** if the member already exists



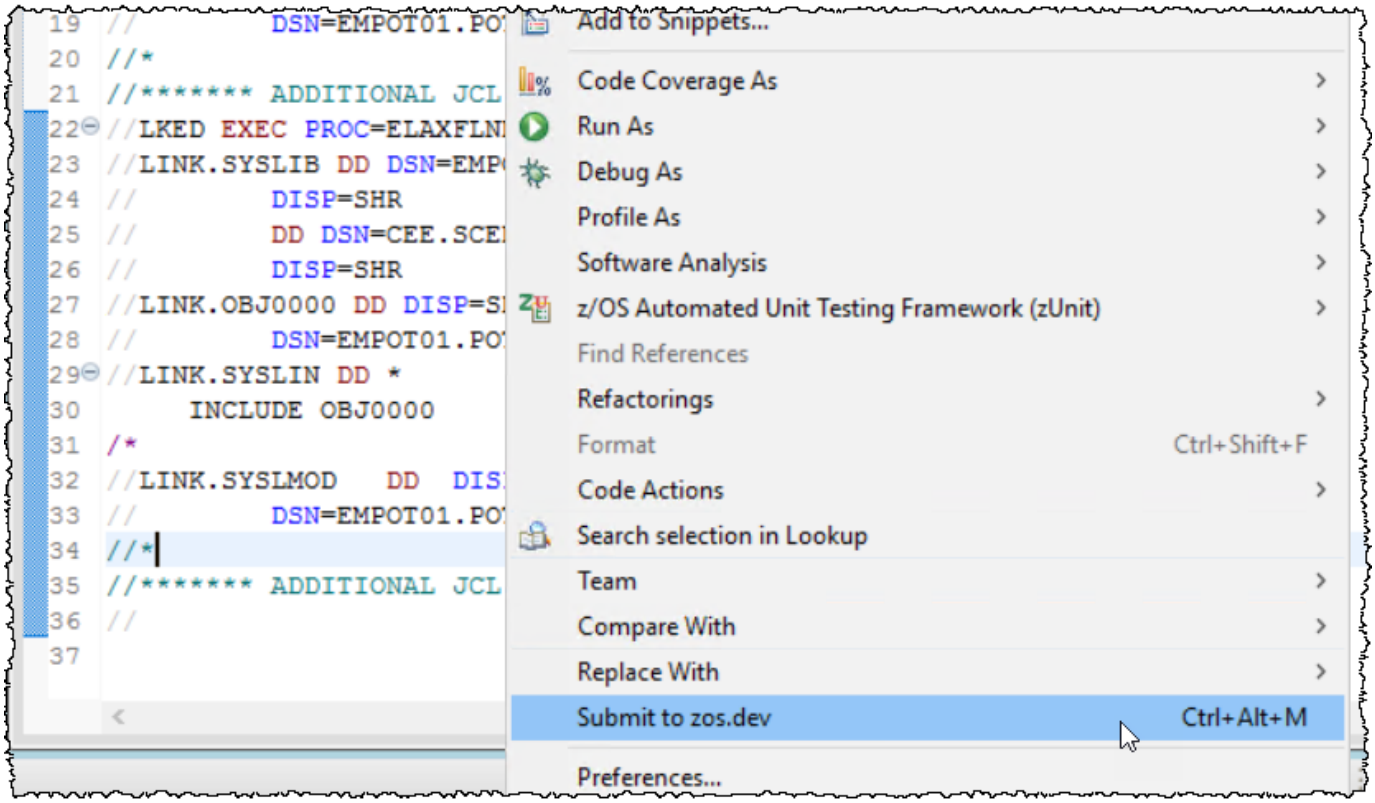
5.3.6 Click **Open** to verify the generated JCL



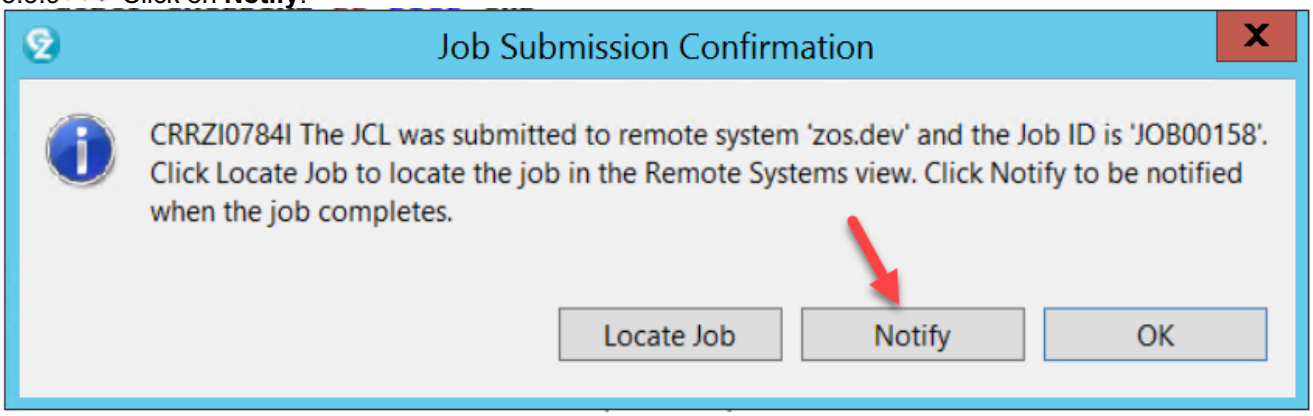
5.3.7 **Scroll down** and notice that the *Load module* will be created on the PDS: **`EMPOT01.POT.LOAD`**.



5.3.8 Right click on the JCL and click **Submit to zos.dev**

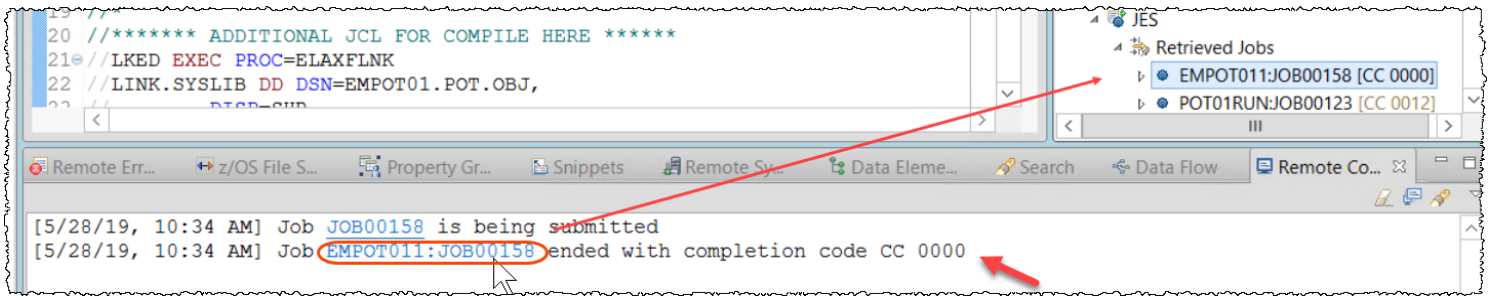


5.3.9 Click on **Notify**.



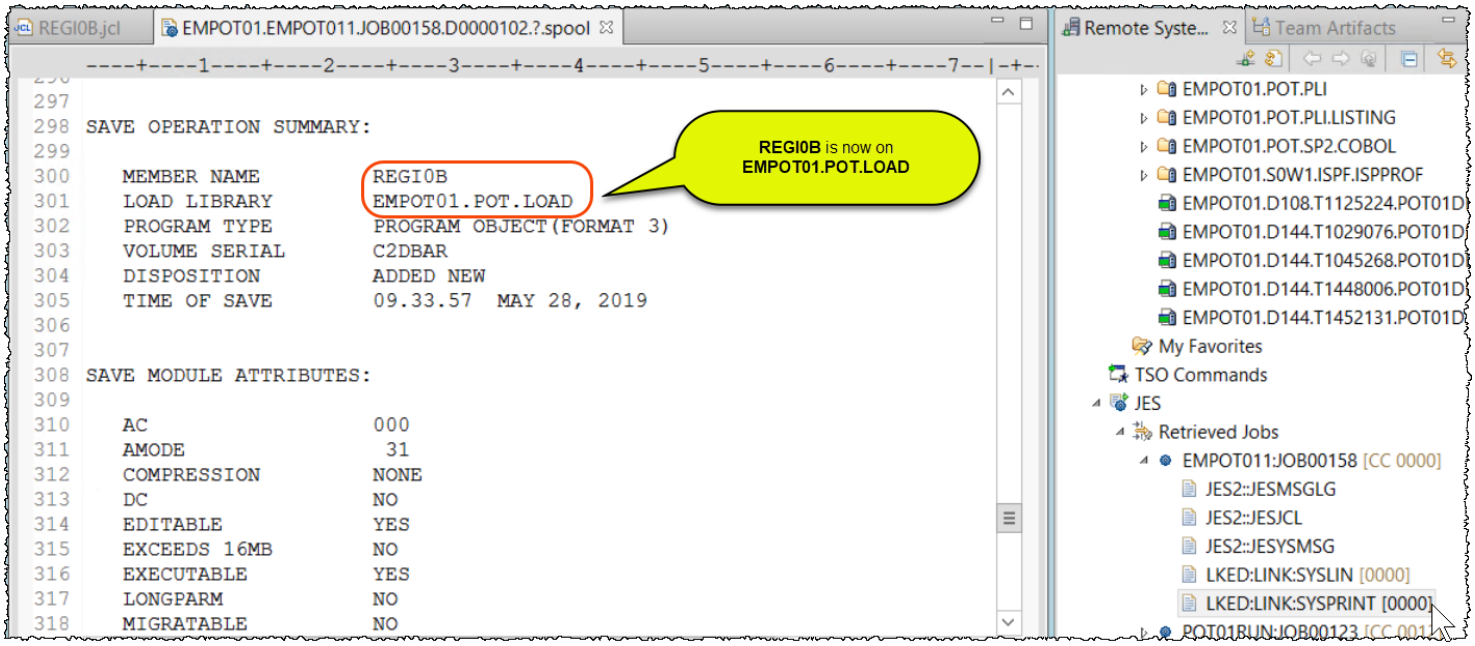
5.3.10 You MUST have **000** as return code.

Click on **EMPOT011:JOB00xxx**



5.3.11 Expand **EMPOT01:JOB00xxx** and verify the results double clicking on **LKED:LINK:SYSPRINT.**

Scroll down and verify that the load module **REGI0B** invoked by your DB2 COBOL program is now created in **EMPOT01.POT.LOAD.**



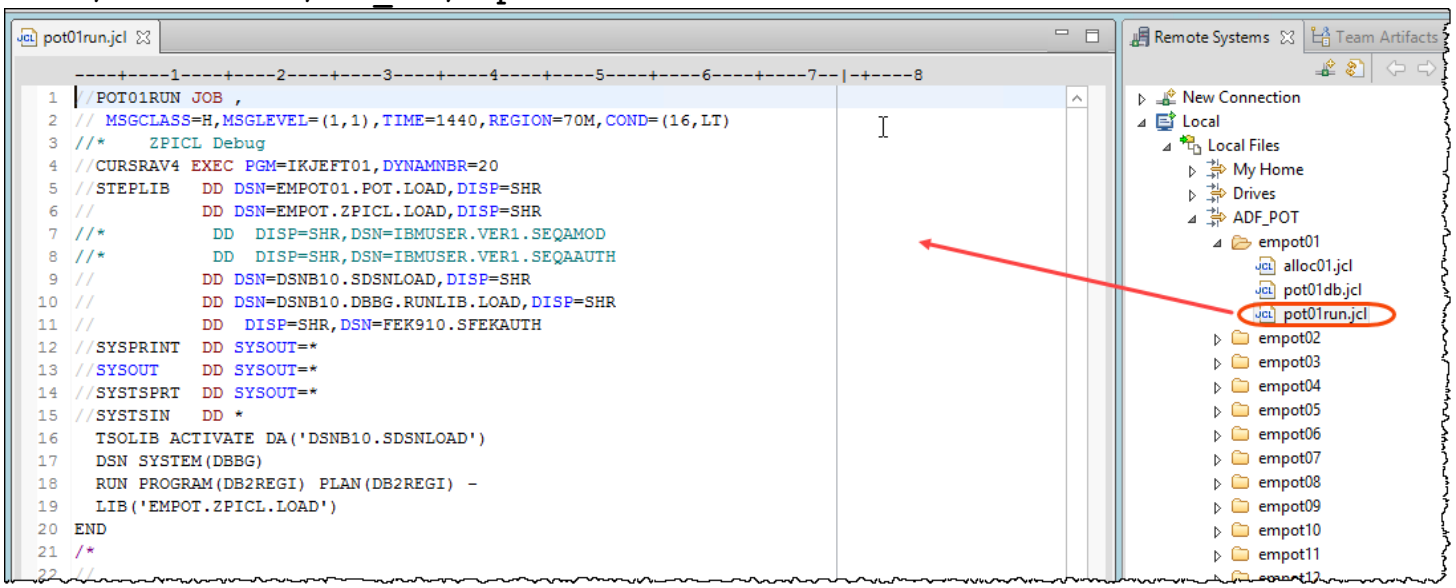
5.3.10 Use **Ctrl + Shift + F4** to close all editors.

Or just click on the . of each opened editor

___ 5.4 Execute the COBOL/DB2 program

On this step, you will explore some capabilities of the JCL editor and modify the JCL for execution.

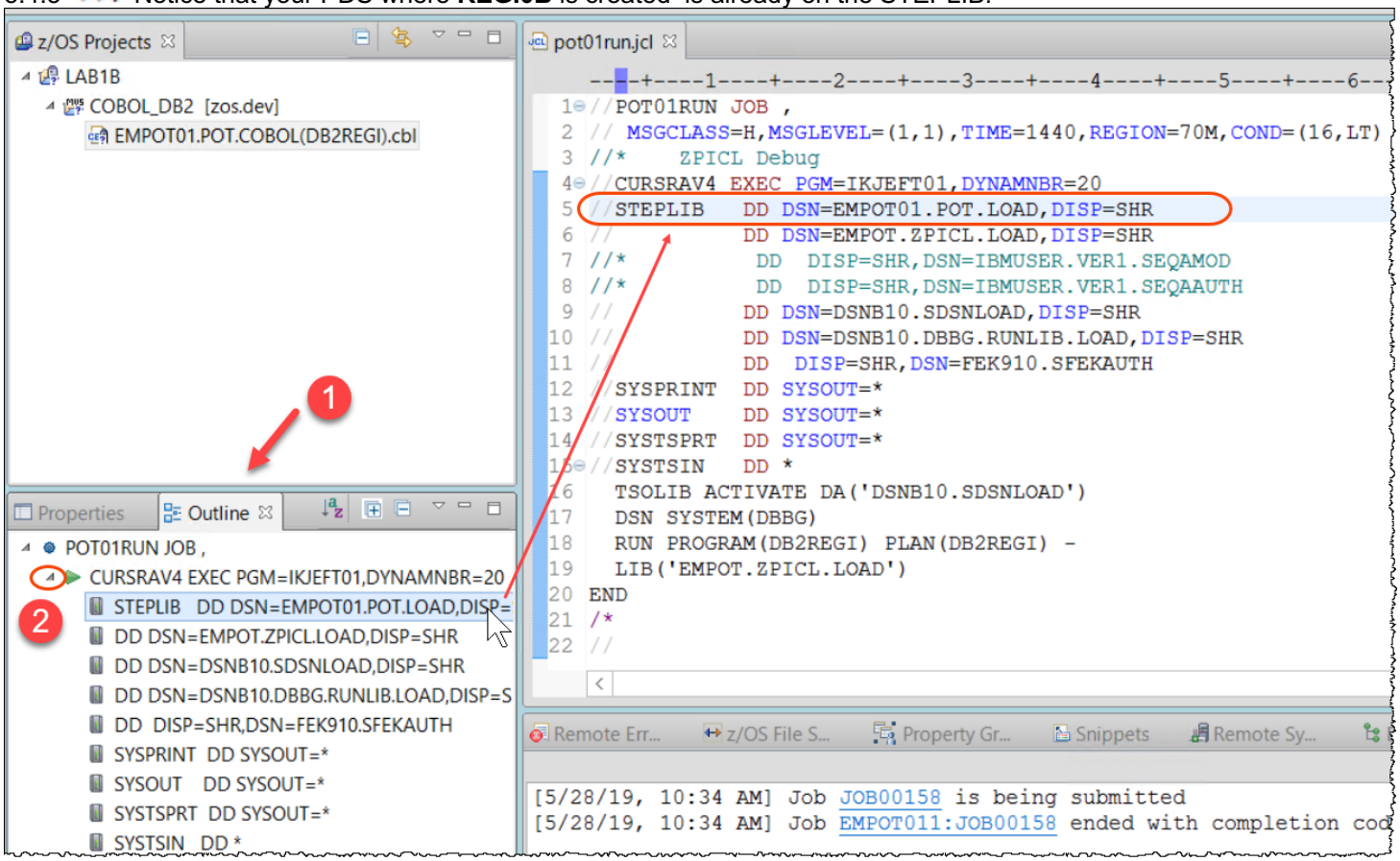
5.4.1 ⏭ Using *Remote System View,* scroll back to locate the file **`pot01run.jcl`** under the folder **`Local/Local Files/ADF_POT/empot01`** and **double click** to edit..



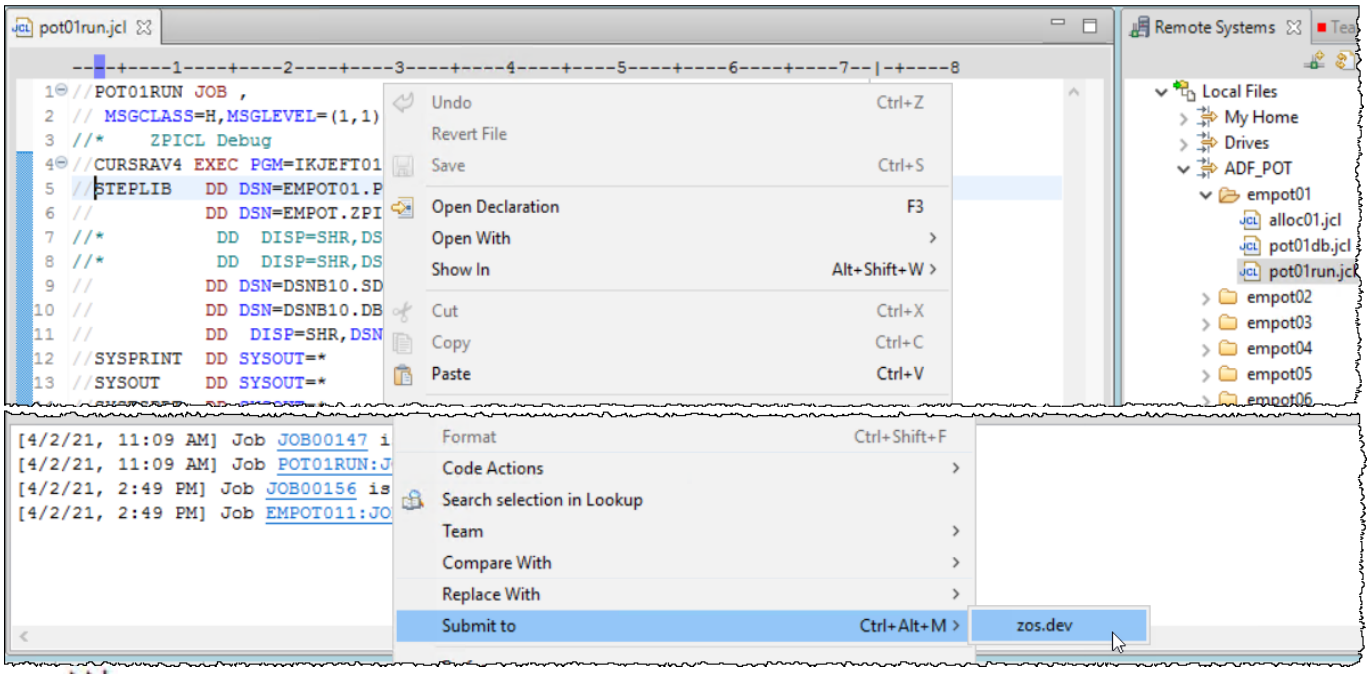
5.4.2 Notice on bottom left corner the same *Outline* view that you have for the COBOL program. It will help to navigate on large JCL members.

⏭ Expand **`CURSRAV4 EXEC PGM=`** and click on **`STEPLIB`**

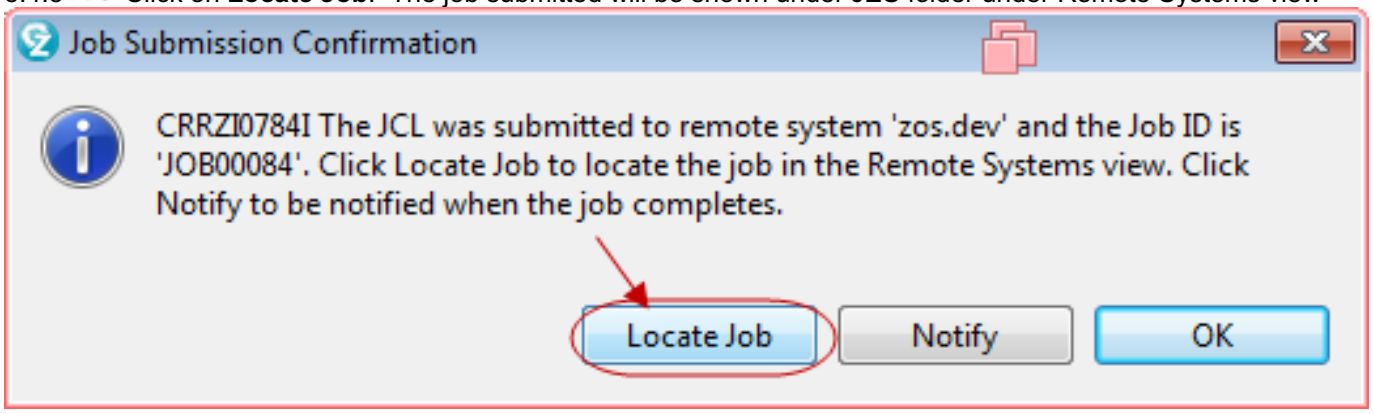
5.4.3 Notice that your PDS where **REGI0B** is created is already on the STEPLIB.



5.4.4 Right click and select **Submit to > zos.dev**



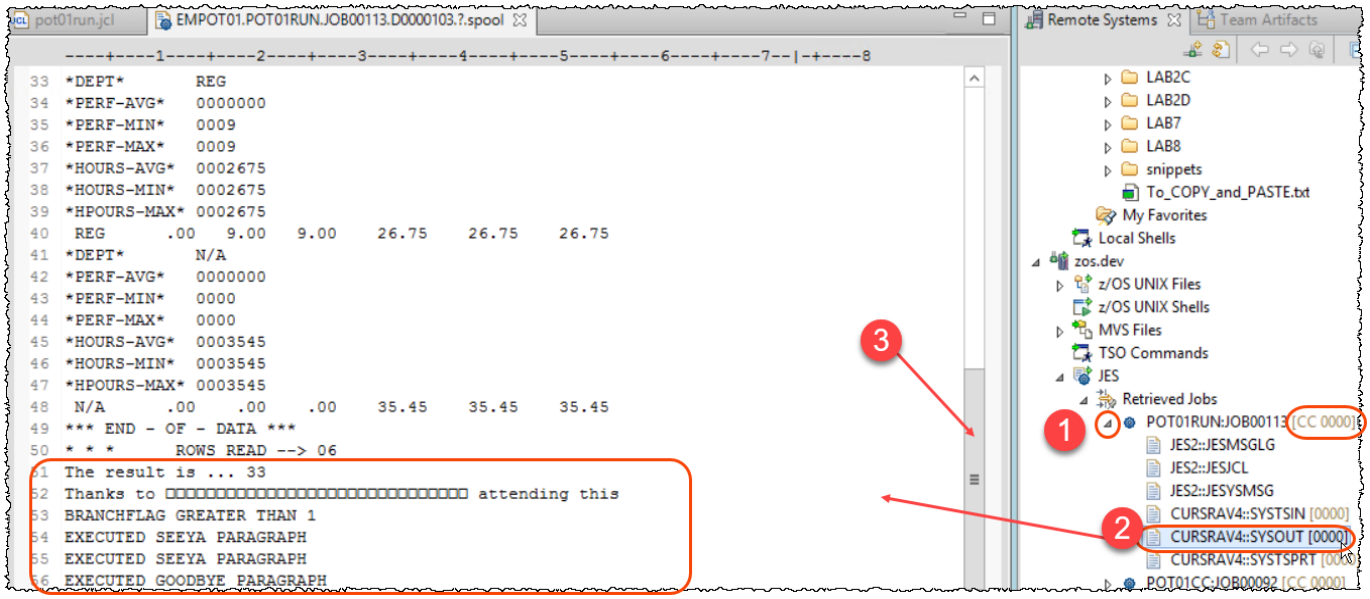
5.4.5 Click on **Locate Job**. The job submitted will be shown under JES folder under Remote Systems view



5.4.6 You MUST have the return code 0000 and the bug will be corrected.

Using *Remote Systems* view, verify the results double expanding **`POT01RUN:JOB00xxx`** (where **`00xxx`** can be any number) and clicking on **`CURSRAV4:SYSOUT.`** Now you MUST have 000 as return code

Tip: You may need to refresh Retrieved Jobs to see it go from active to finished


pot01run.jcl
EMPOT01.POT01RUN.JOB00113.D0000103.?.spool
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
33 *DEPT*   REG
34 *PERF-AVG*   0000000
35 *PERF-MIN*   0009
36 *PERF-MAX*   0009
37 *HOURS-AVG* 0002675
38 *HOURS-MIN* 0002675
39 *HPOURS-MAX* 0002675
40 REG   .00   9.00   9.00   26.75   26.75   26.75
41 *DEPT*   N/A
42 *PERF-AVG*   0000000
43 *PERF-MIN*   0000
44 *PERF-MAX*   0000
45 *HOURS-AVG* 0003545
46 *HOURS-MIN* 0003545
47 *HPOURS-MAX* 0003545
48 N/A   .00   .00   .00   35.45   35.45   35.45
49 *** END - OF - DATA ***
50 * * *   ROWS READ --> 06
51 The result is ... 33
52 Thanks to [redacted] attending this
53 BRANCHFLAG GREATER THAN 1
54 EXECUTED SEEYA PARAGRAPH
55 EXECUTED SEEYA PARAGRAPH
56 EXECUTED GOODBYE PARAGRAPH
Remote Systems
Team Artifacts
LAB2C
LAB2D
LAB7
LAB8
snippets
To_COPY_and_PASTE.txt
My Favorites
Local Shells
zos.dev
z/OS UNIX Files
z/OS UNIX Shells
MVS Files
TSO Commands
JES
Retrieved Jobs
POT01RUN:JOB00113 [CC 0000]
JES2::JESMSGLG
JES2::JESJCL
JES2::JESYSMSG
CURSRAV4::SYSTSIN [0000]
CURSRAV4::SYSOUT [0000]
CURSRAV4::SYSTSPRT [0000]
POT01CC:JOB00092 [CC 0000]
1
2
3

5.4.7 Use **Ctrl + Shift + F4** to close all editors. Or just click on the . of each opened editor

What have you done so far?
You used **Fault Analyzer** to identify why the program was abending.
You used the **Debugger** to temporarily fix the error caused by the called program REGI0B.
You used IDz to change the program REGI0B to return other value than zero.
You compiled and linked REGI0B creating another version on your load library.
You executed again the JOB but now using the REGI0B that you created and verified that the abend is gone.

Section 6. Using the Code coverage

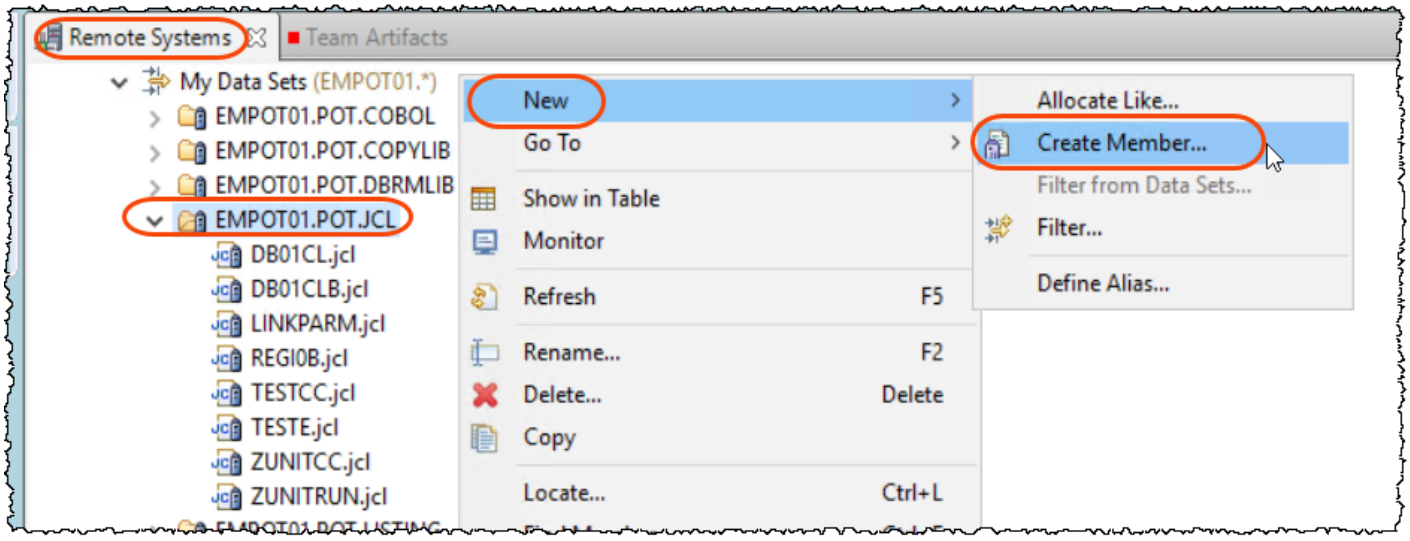
Code coverage analyzes a running program and generates a report of statements that were executed, compared to the total number of executable lines.
Developer for z Systems Host Utilities provides two ways to invoke Code coverage in batch mode, A sample JCL procedure, to process a single program run, and a set of scripts to start and stop a permanently active Code coverage collector that can process multiple program runs.

You can run code coverage for any application you can debug. You can generate code coverage reports that you can view in the workbench or save the results to your file system for future analysis. We will show a simple batch mode invocation.

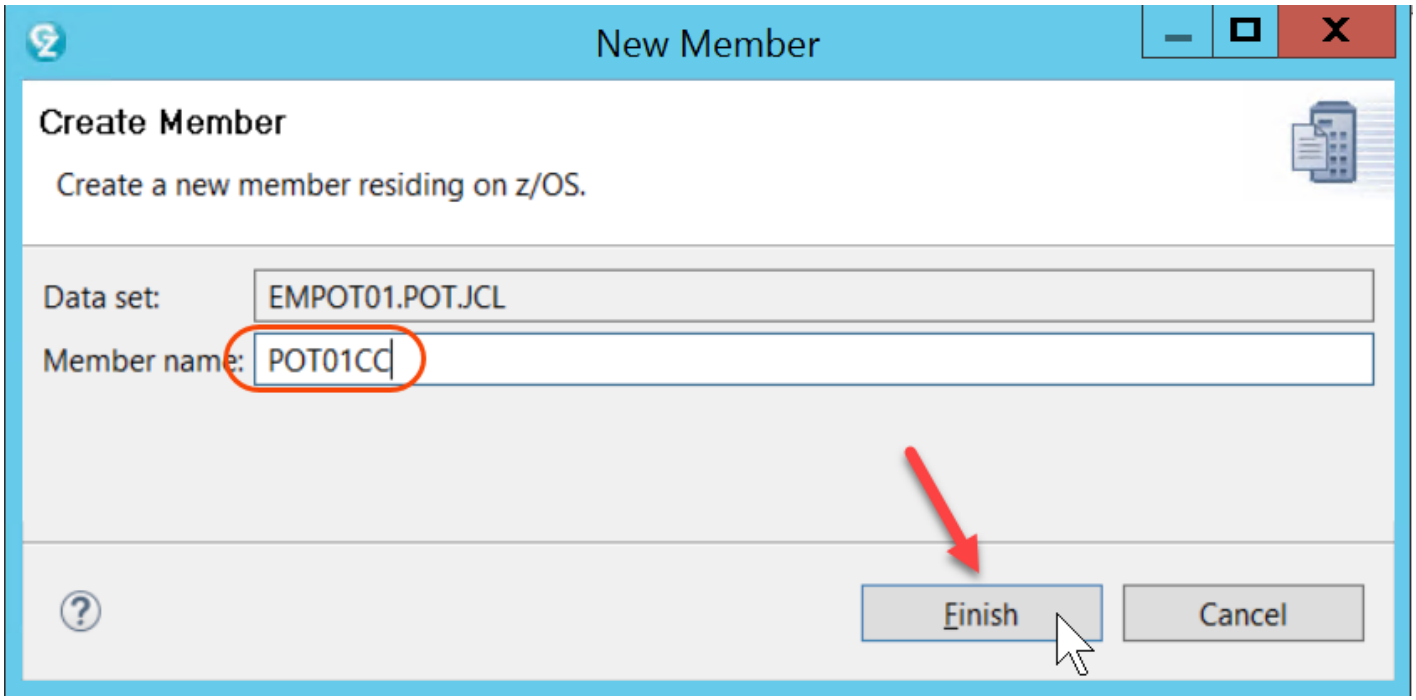
___ 6.1 Creating a JCL to run the code coverage

You will create a JCL file to run Code Coverage

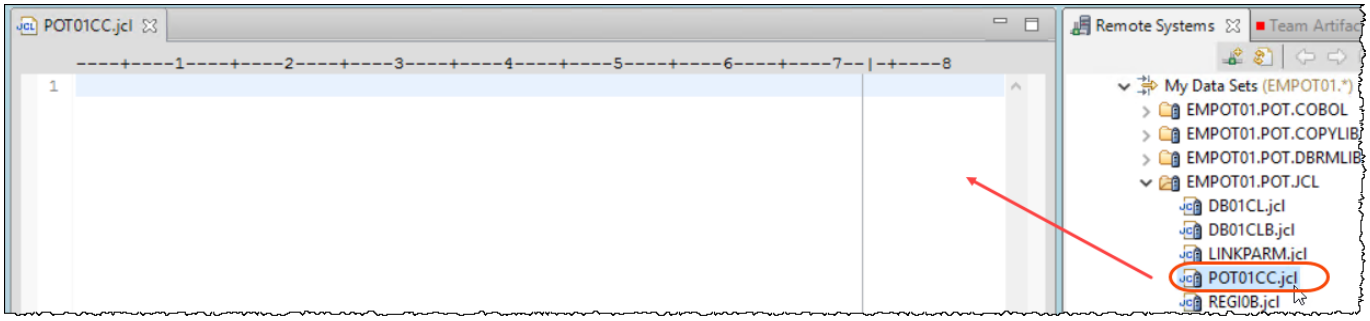
6.1.1 Using Remote Systems view, right click on **EMPOT01.POT.JCL** and select **New > Create Member…**



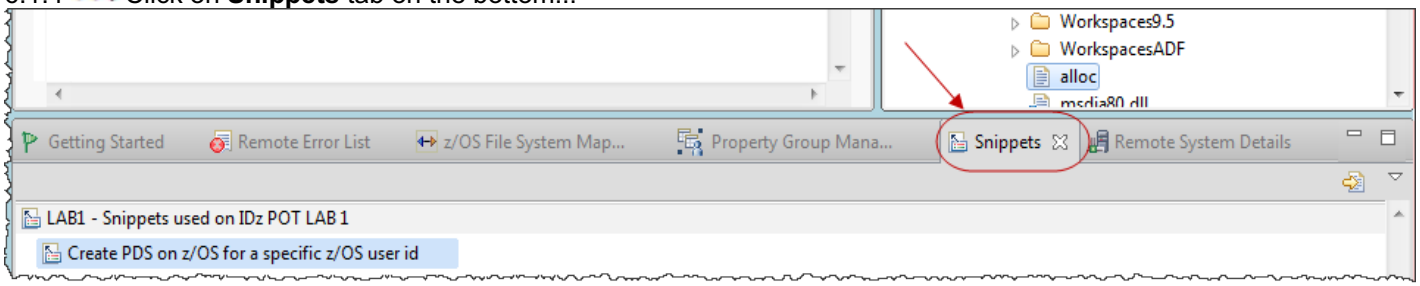
6.1.2 Name it as **POT01CC** and click **Finish**


New Member
Create Member
Create a new member residing on z/OS.
Data set:
EMPOT01.POT.JCL
Member name:
POT01CC
Finish
Cancel

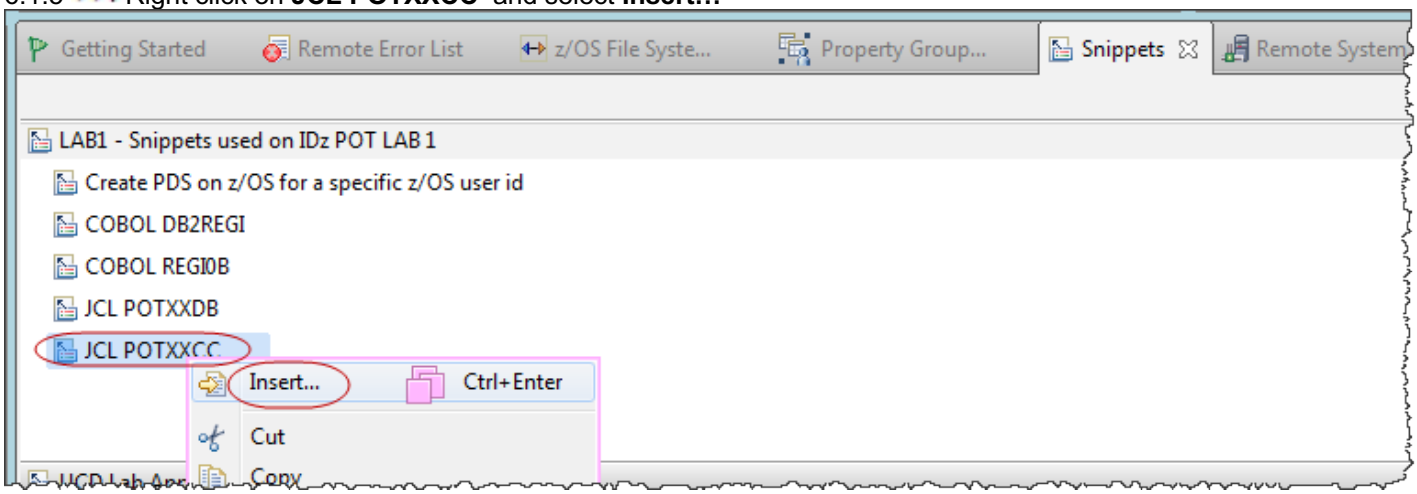
6.1.3 Double click on **POT01CC** to edit it



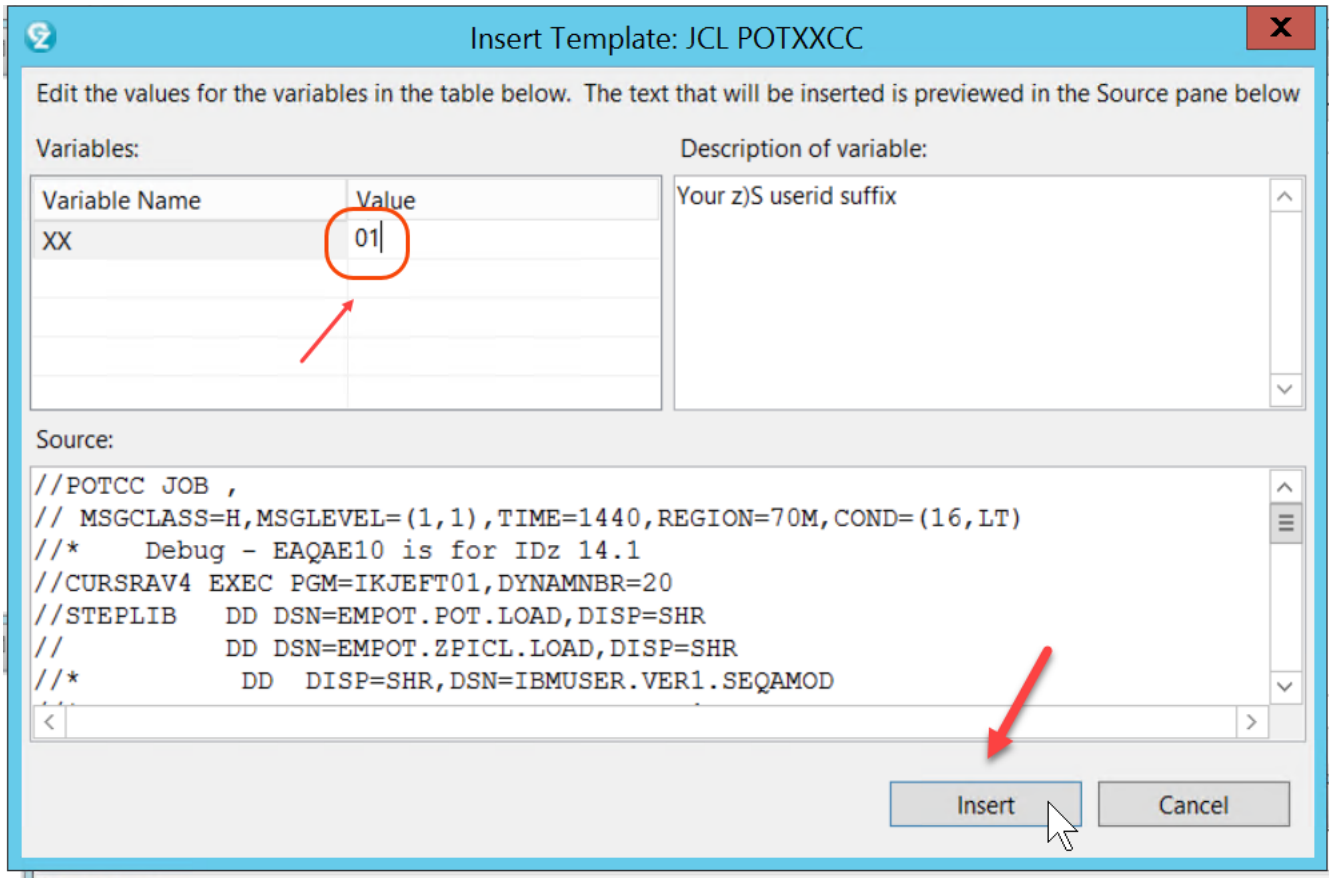
6.1.4 Click on **Snippets** tab on the bottom...



6.1.5 Right click on **JCL POTXXCC** and select **Insert…**

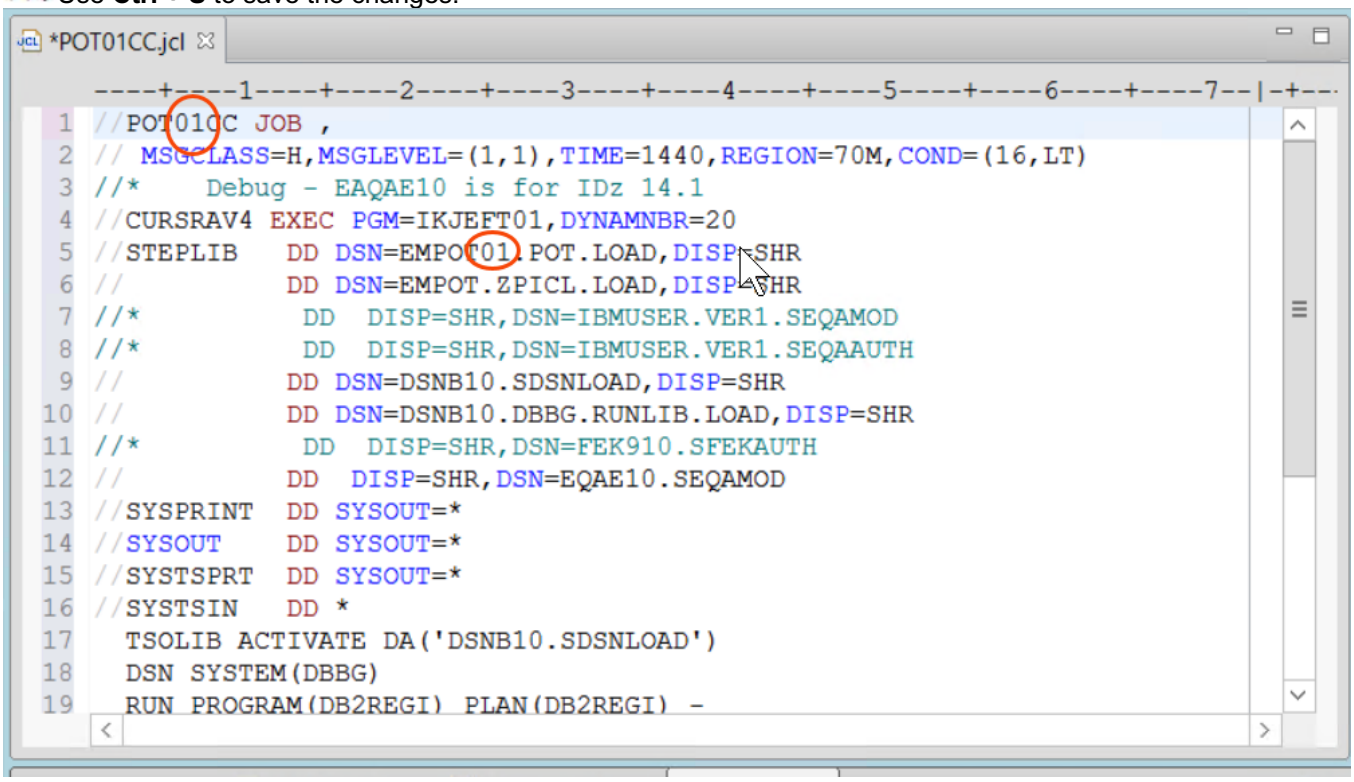


6.1.6 When the *Insert* dialog opens, type **01** in the *Value* and click **Insert**


Insert Template: JCL POTXXCC
Edit the values for the variables in the table below. The text that will be inserted is previewed in the Source pane below
Variables:
Variable Name
Value
XX
01
Description of variable:
Your z)S userid suffix
Source:
//POTCC JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
//* Debug - EAQAE10 is for IDz 14.1
//CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DSN=EMPOT.POT.LOAD,DISP=SHR
// DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR
//* DD DISP=SHR,DSN=IBMUSER.VER1.SEQAMOD
Insert
Cancel

6.1.7 . Notice that the Snippets variables will be replaced.

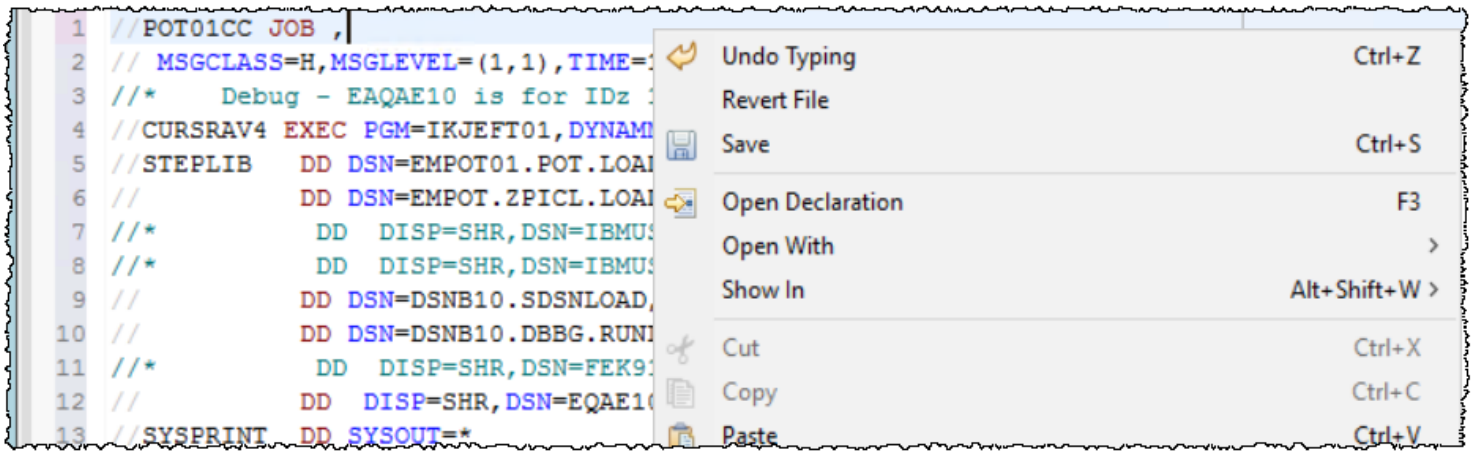
Use **Ctrl + S** to save the changes.

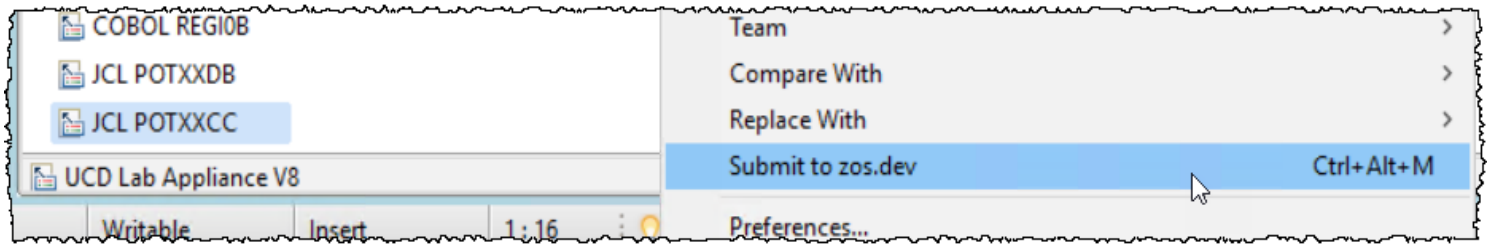


___ 6.2 Submit the JCL for z/OS execution

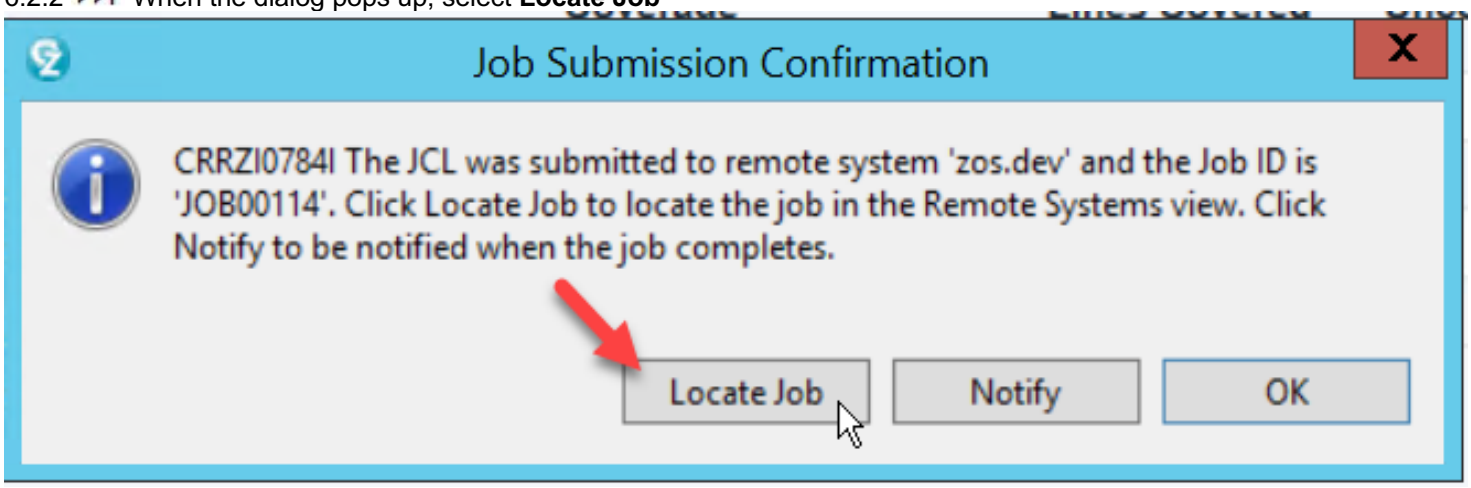
You will now execute the fixed batch COBOL/DB2 on z/OS

6.2.1 **Right click** on the JCL being edited and select **Submit to zos.dev**



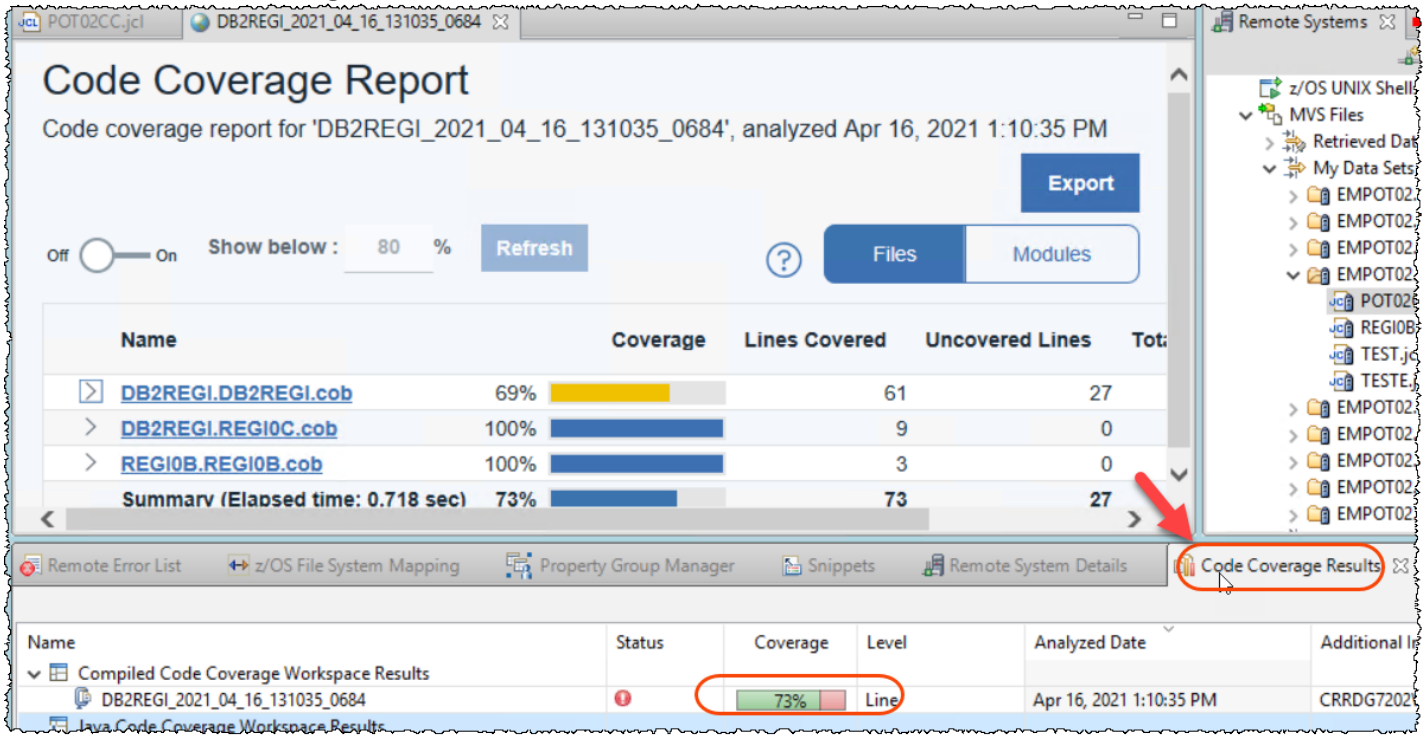


6.2.2 When the dialog pops up, select **Locate Job**



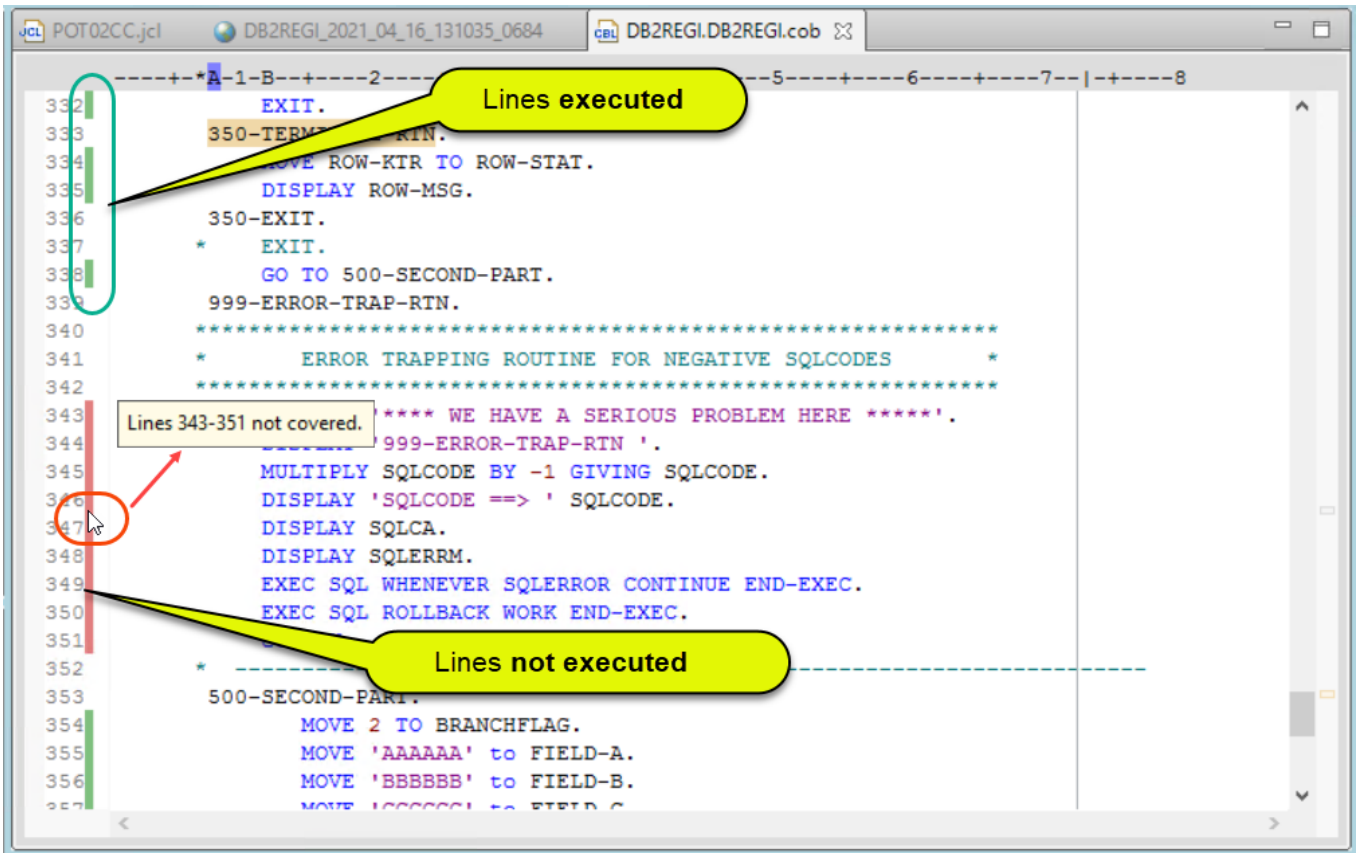
6.2.3 The code coverage report will be created.You can see that this test covered only **73%** of your program

⏭ Click on **Code Coverage Results** tab



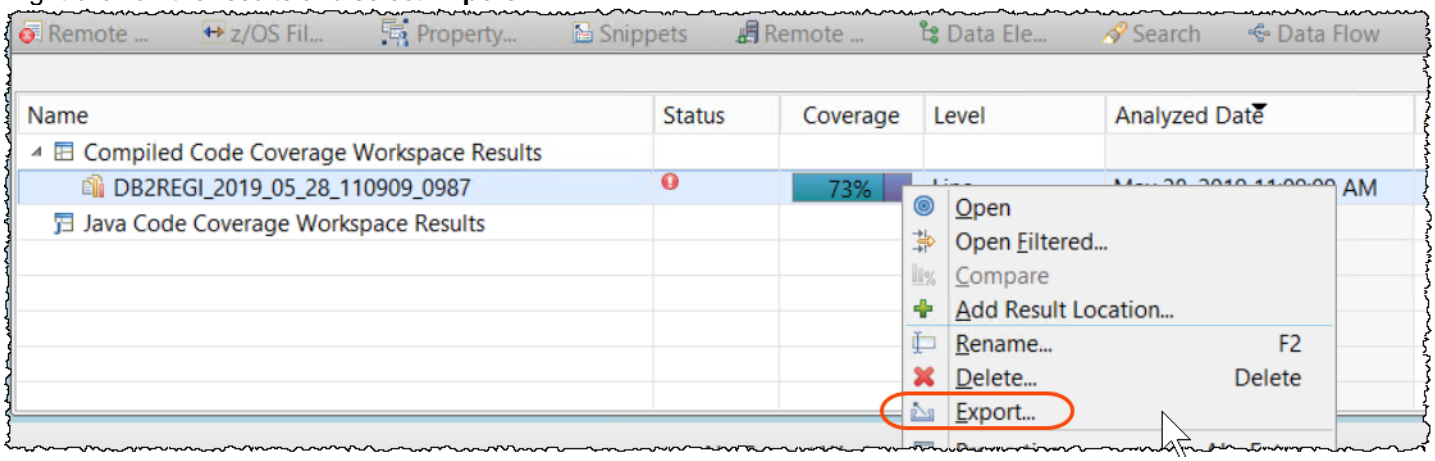
6.2.4 ⏭ Click on **DB2REGI.DB2REGI.cob**

6.2.5 ⏭ Scroll down move the mouse to the colored green and red bars
and you will see the lines executed in **green** and the lines not executed in **Red**.

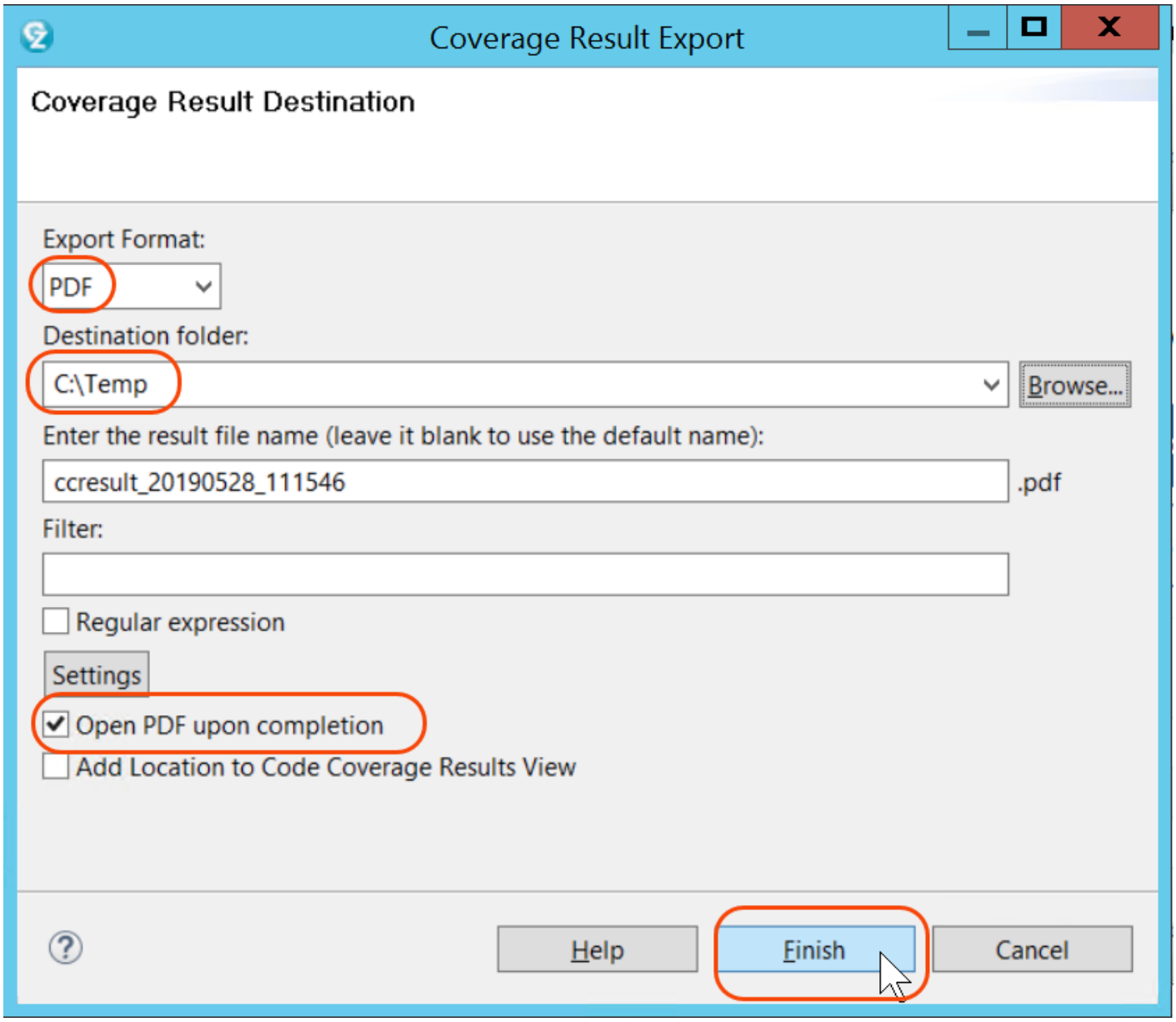


As you can see the error routine on the picture above is not tested..
Imagine how bad if you go to production and never test this condition ....

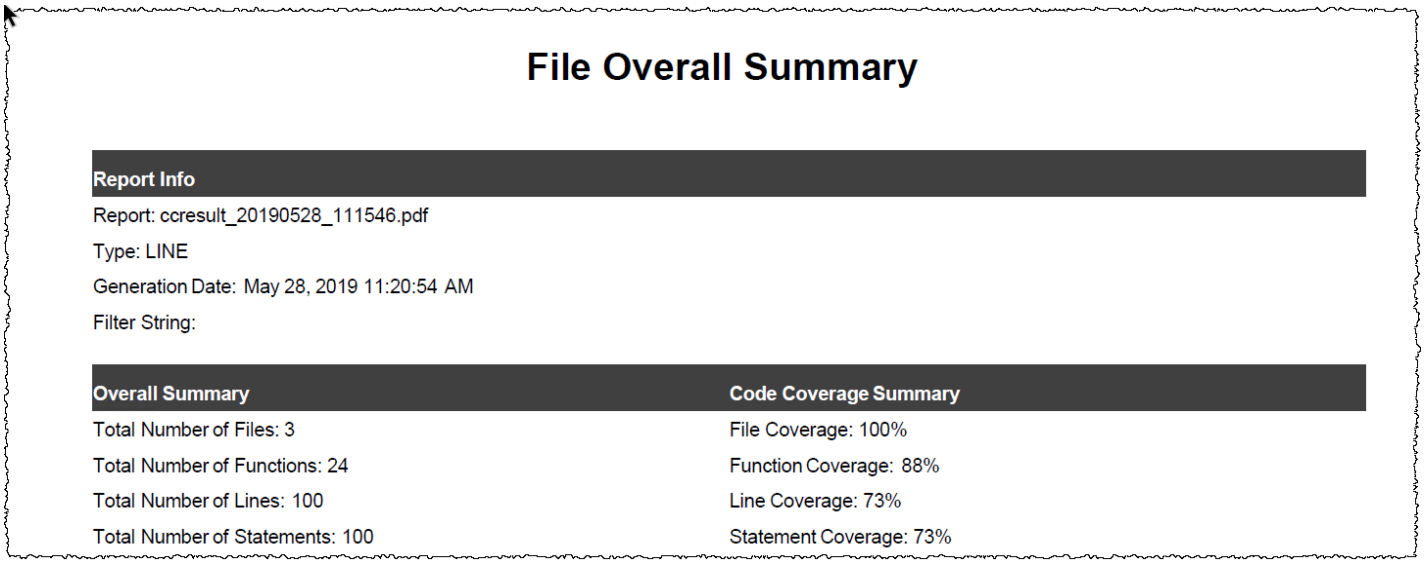
6.2.6 You could also create a PDF report as documentation or export this data to other products (like ADDI or SonarQube)
Right click on the results and select **Export** ..



6.2.7 Select **PDF**, **C:\Temp, Open PDF upon completion** and **Finish**.
Note that you can export to **Sonar Qube** if you have this software.



6.2.8 A PDF report will be generated.



File Overall Summary

Report Info
Report: ccresult_20190528_111546.pdf
Type: LINE
Generation Date: May 28, 2019 11:20:54 AM
Filter String:

Overall Summary	Code Coverage Summary
Total Number of Files: 3	File Coverage: 100%
Total Number of Functions: 24	Function Coverage: 88%
Total Number of Lines: 100	Line Coverage: 73%
Total Number of Statements: 100	Statement Coverage: 73%

6.2.8 Close all editors pressing **Ctrl + Shift + F4** Or just click on the . of each opened editor**.**

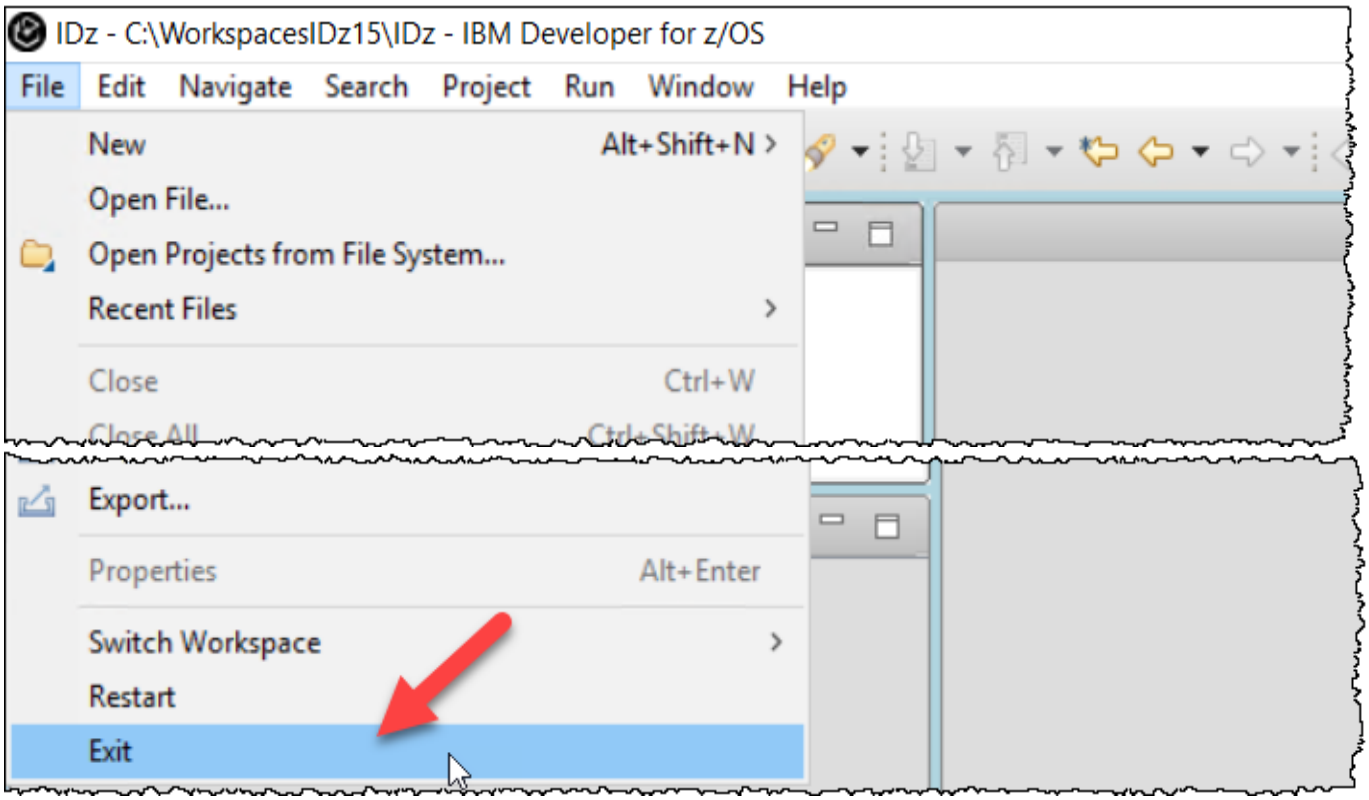
Section 7. (Optional) Executing SQL statements when editing the program.

IDz can be installed with Data studio that is very helpful when working with a Database.
Data Studio is a foundational offering that includes support for key tasks across the data management lifecycle, including administration, application development, and query tuning.

___ 7.0 Starting IDz version 14 and connect to z/OS

When we created this lab Data studio could be installed only on IDz V 14. So for this section you will use IDz Version 14 instead of IDz version 15. Notice that **Data Studio** is a free plugin.

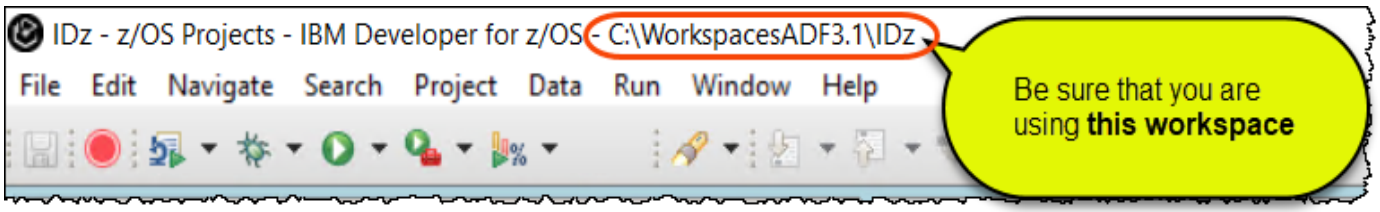
7.0.1 Before starting IDz version 14, you must close IDz version 15 to save some memory on your windows client.
⏭ Click on **File > Exit**



7.0.2 Start ***IBM Developer for z Systems version 14***
⏭ Using the windows desktop double click on **IDz 14.2.4** icon.

IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon.

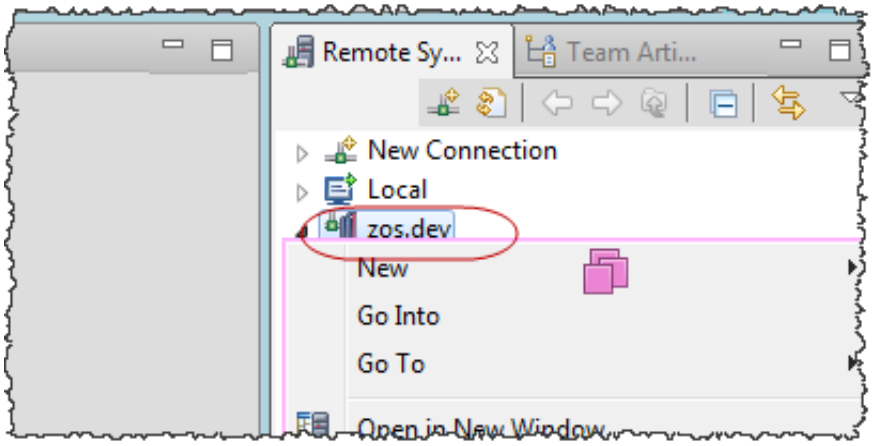

IBM Developer for z/OS
Launching, please wait ...
IBM® Data Studio Version 4.1.3.6 (Aqua Release)
IMS Enterprise Suite Explorer for Development 3.3.1.22
IBM Developer for z/OS
Loading com.ibm.etools.fm.core
Licensed Materials - Property of IBM Corp.© IBM Corporation and other(s) 1998, 2017.
of IBM Corp. in the United States, other countries, or both.
See that **Data Studio** is installed on this IDz version

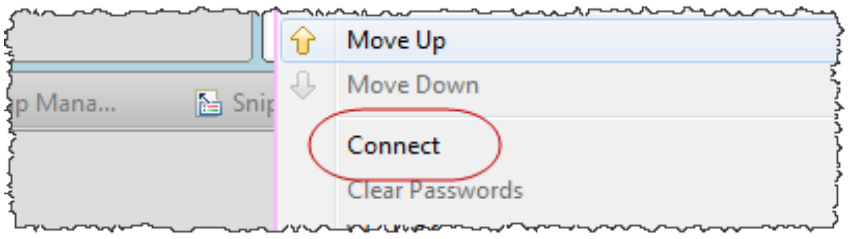

IDz - z/OS Projects - IBM Developer for z/OS - C:\WorkspacesADF3.1\IDz
File Edit Navigate Search Project Data Run Window Help
Be sure that you are using **this workspace**

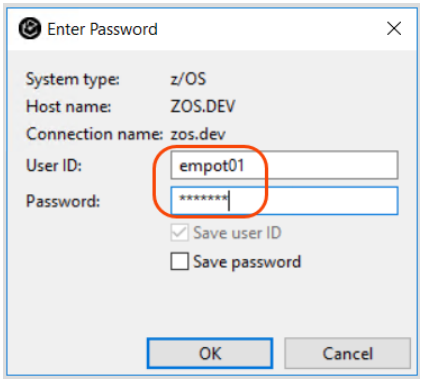
7.0.3 To Connect to the z/OS system:

⏭ Using the *Remote Systems view,* right-click on **zos.dev** and select **Connect .**

⏭ Use **empot01** and password **empot01**



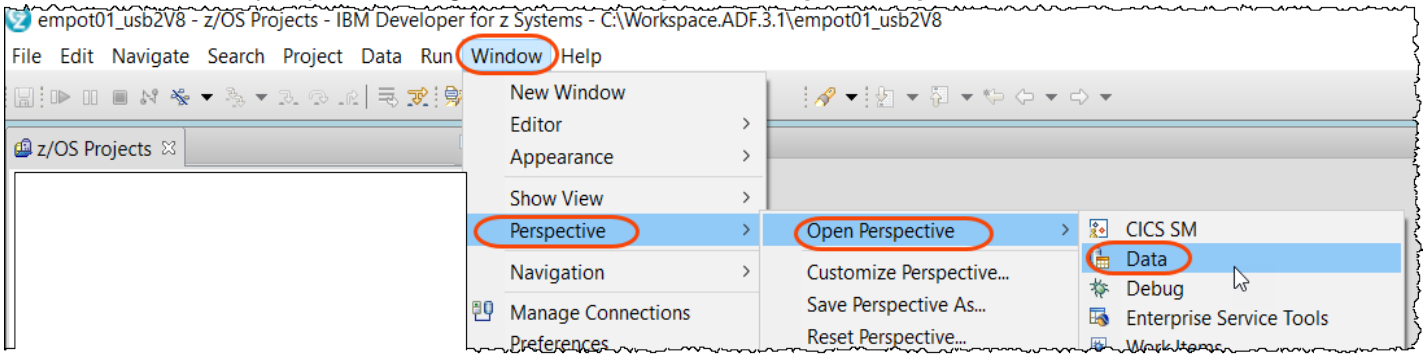




___ 7.1 Creating a connection to the DB2 on z/OS

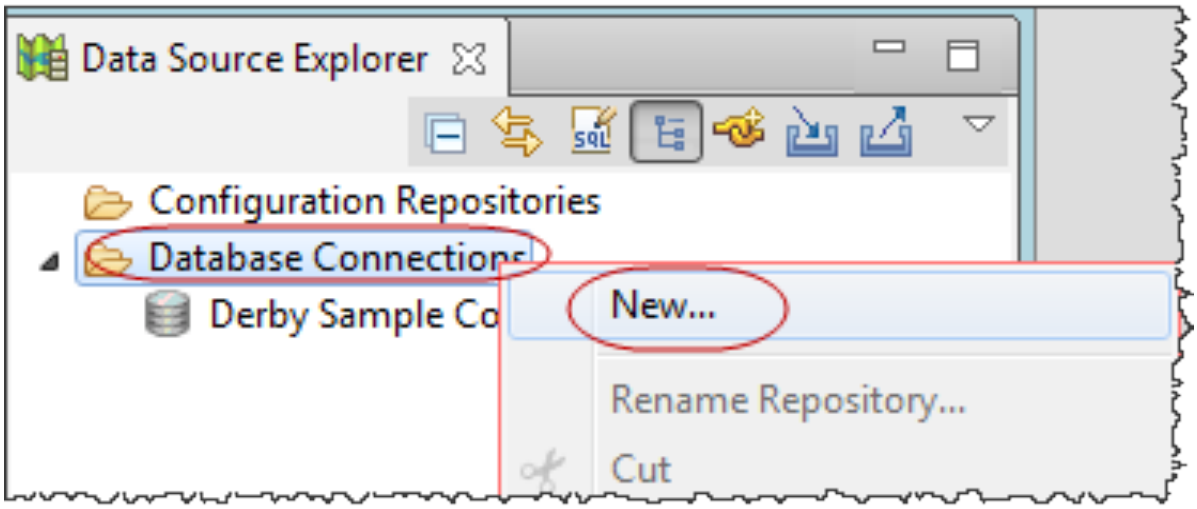
You will need to have authorization to connect to the DB2. Now you will use **IBMUSER** as a new z/OS ID

7.1.1 ⏭ Go to **Data** perspective using **Window > Perspective > Open Perspective > Data**

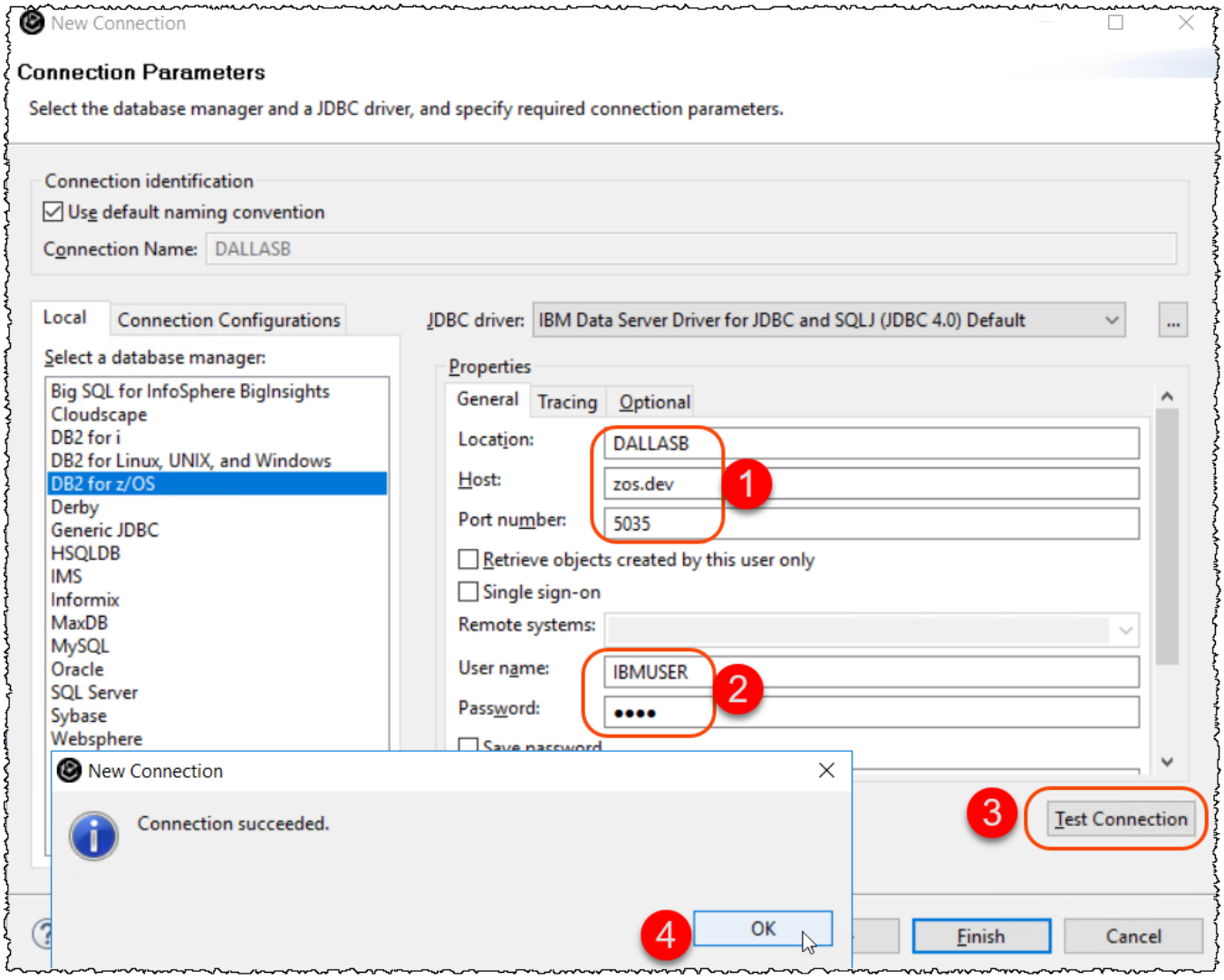


7.1.2Using the **Data Source Explorer** view, create a new z/OS DB2 connection:

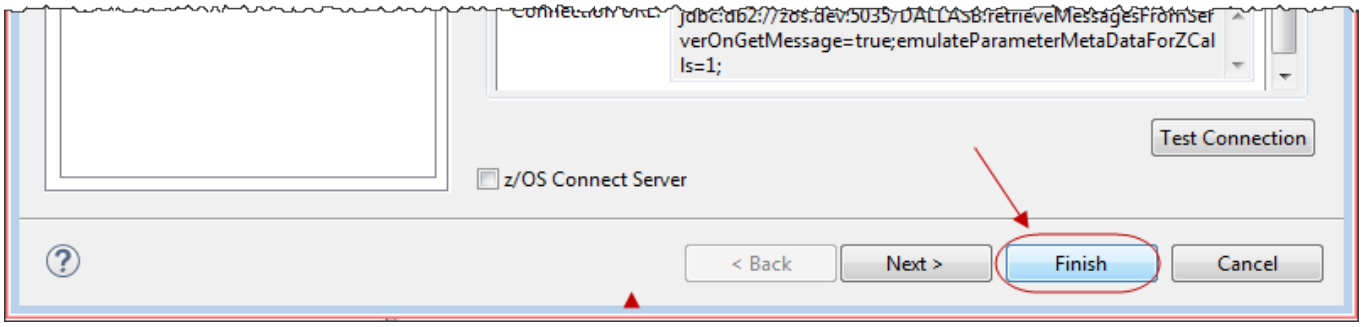
⏭ Right click on **Database Connections** and select **New..**



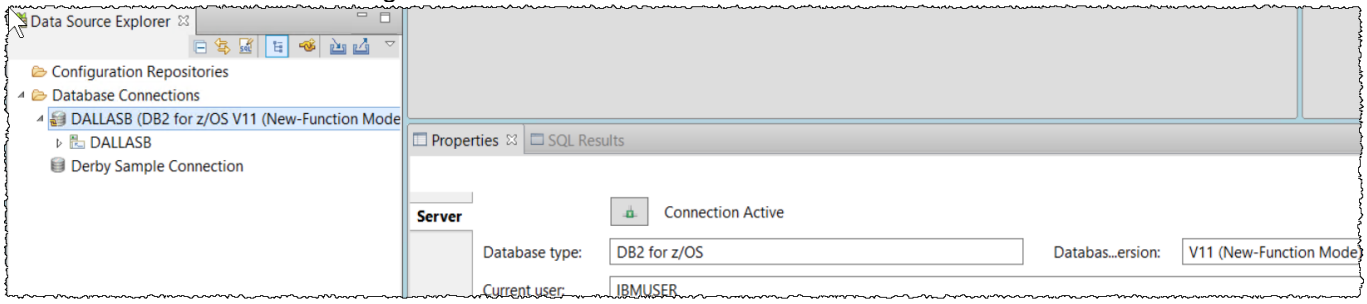
7.1.3 Select **DB2 for z/OS,** use *Location* as **DALLASB**,
Host is **zos.dev** and port is **5035**. **, IBMUSER** and password **SYS1**
Click **Test Connection** to be sure you can successfully connect to the DB2.



7.1.4 Click **OK** and **Finish**



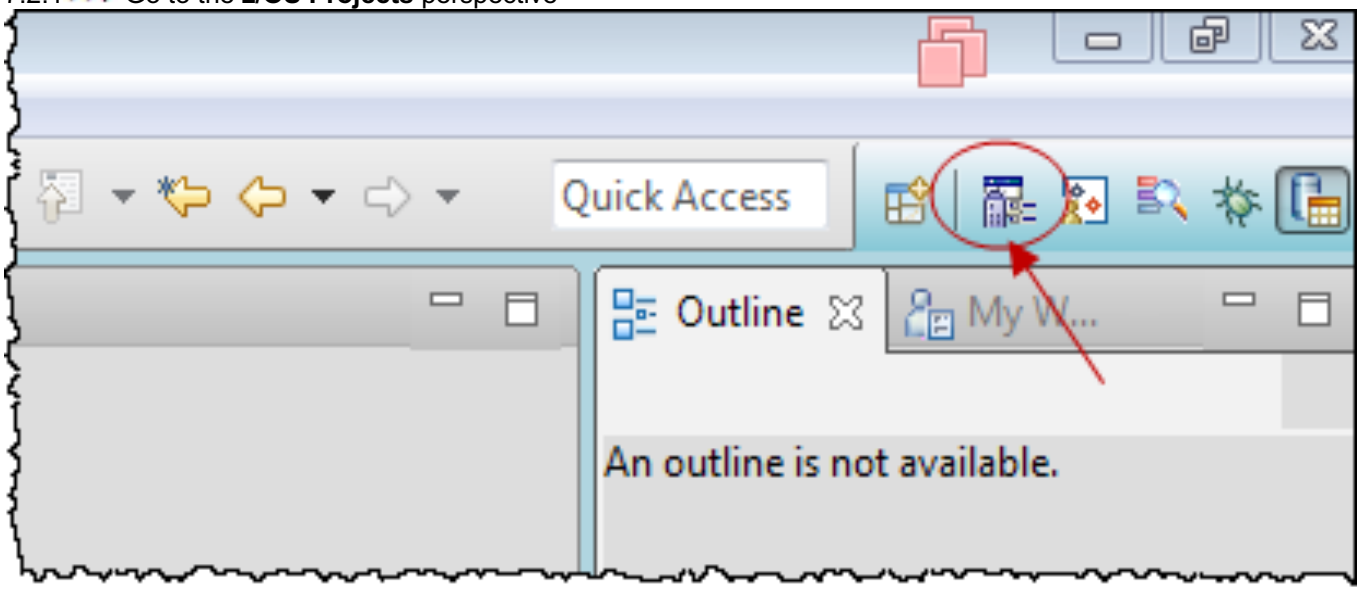
The connection is created. You could see tables, modify contents etc..
But we want to focus on the integration of DB2 and the COBOL/DB2 editor.



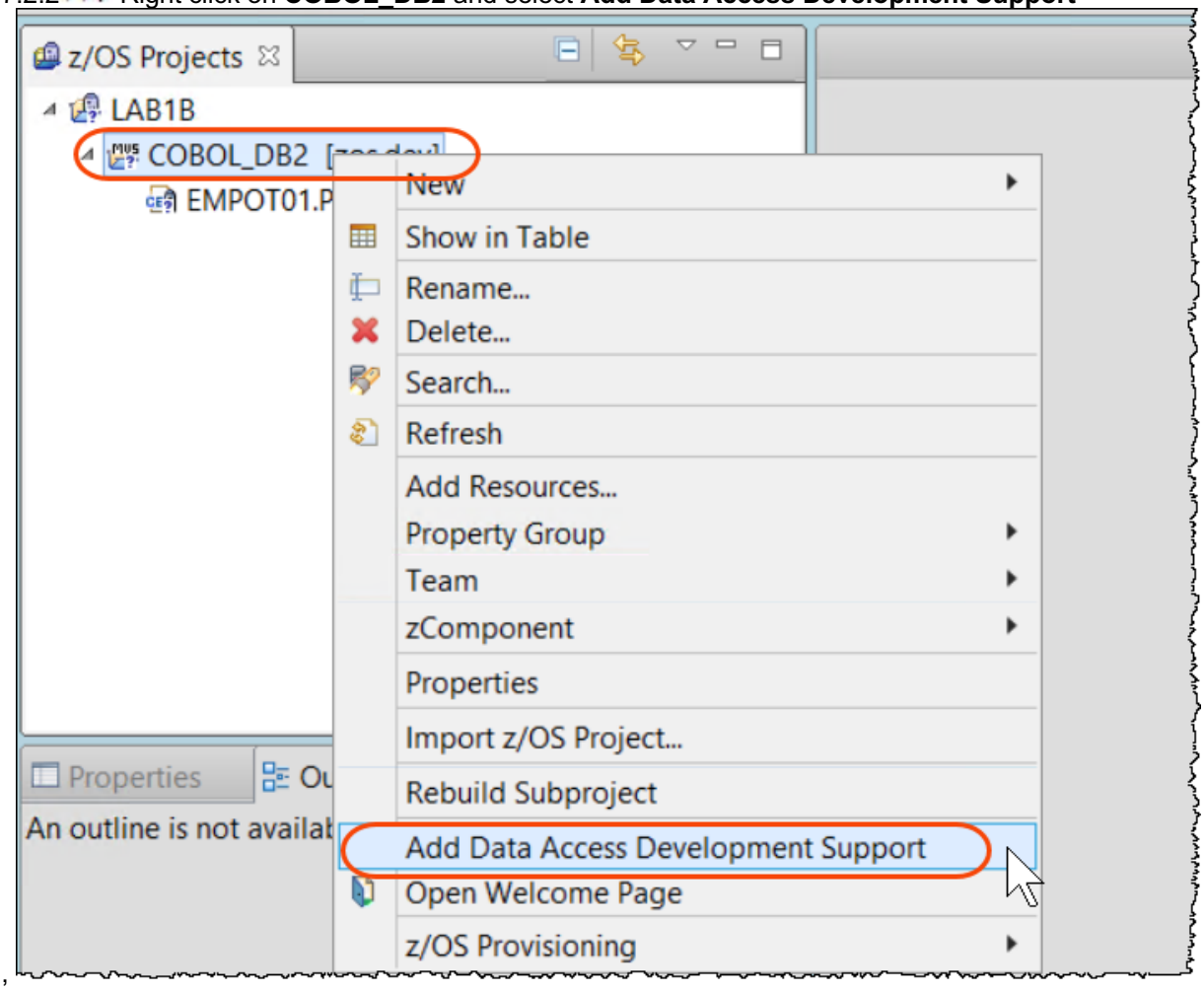
___ 7.2 Running a SQL query from COBOL program

You will need to have authorization to run the queries.

7.2.1 Go to the **z/OS Projects** perspective

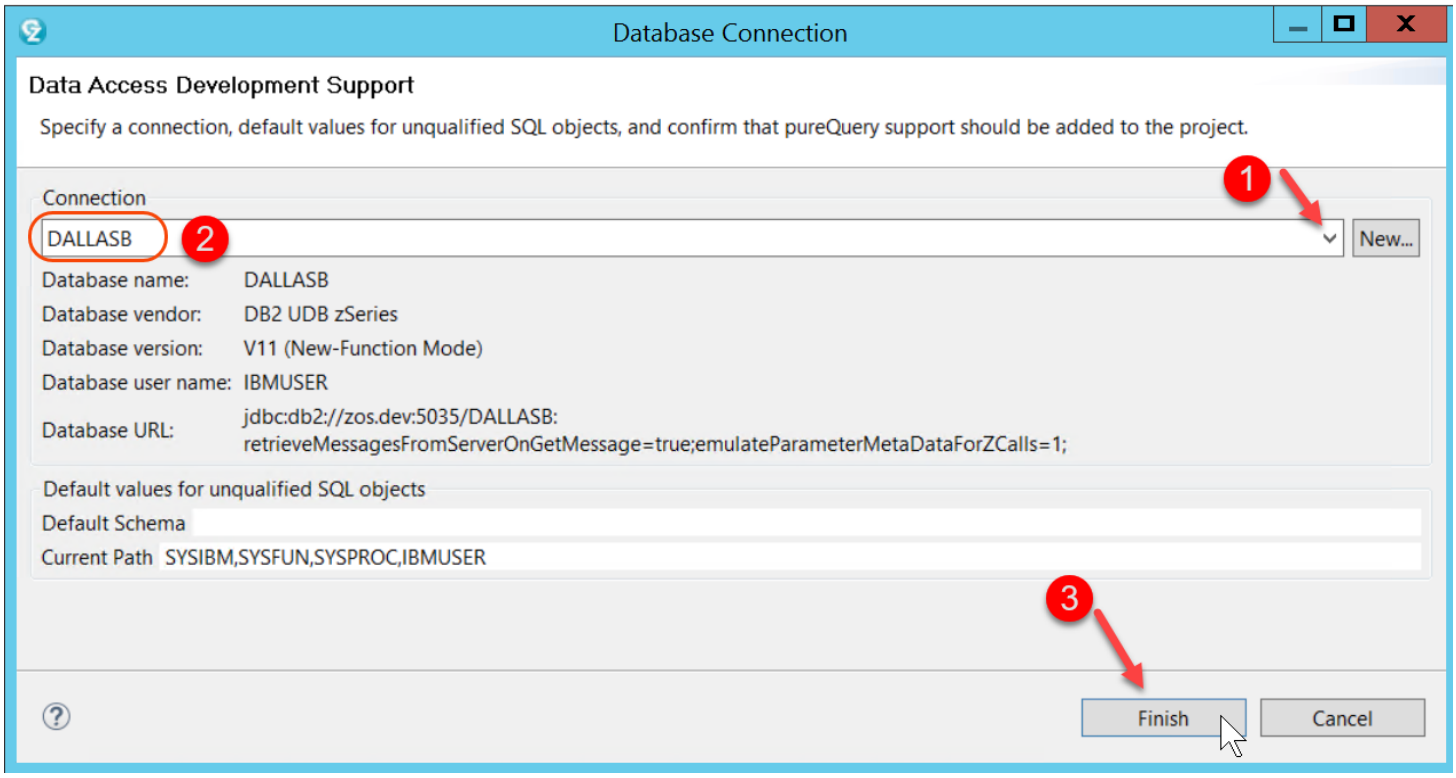


7.2.2 Right click on **COBOL_DB2** and select **Add Data Access Development Support**

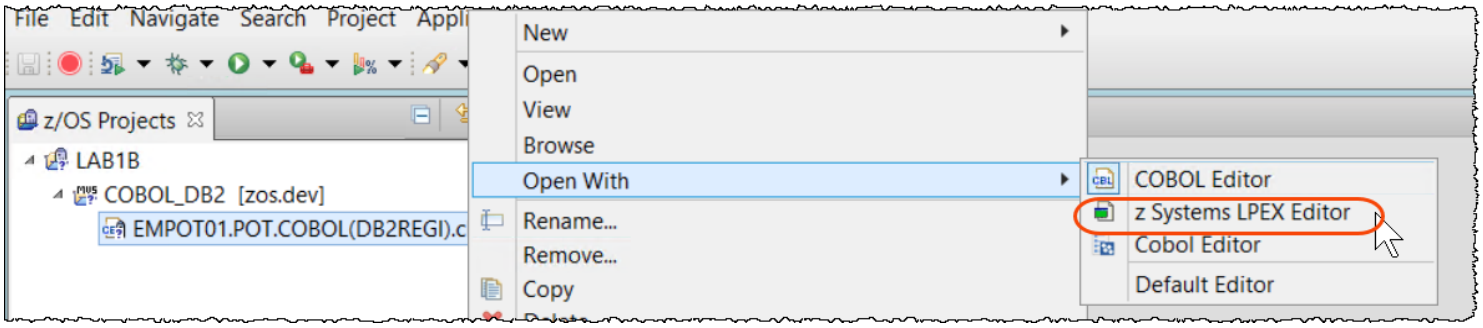


,

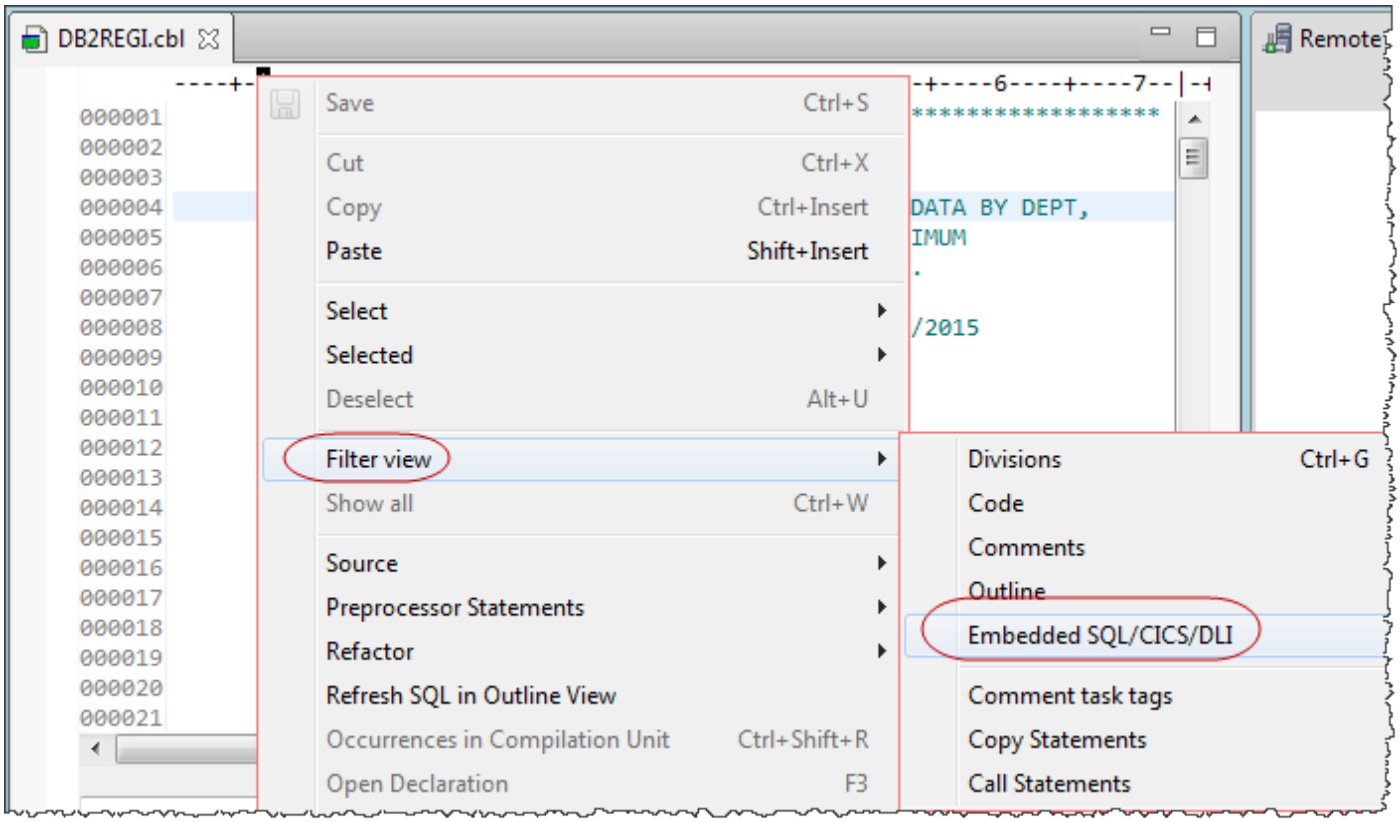
7.2.3 Select the connection created **DALLASB** and click **Finish**



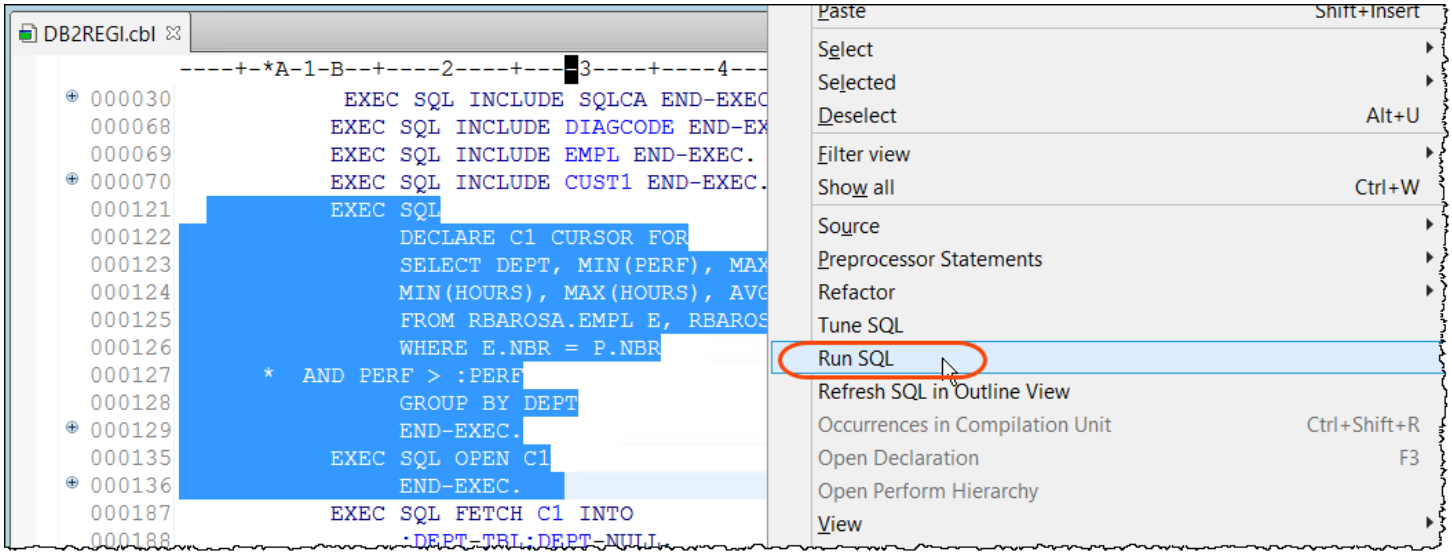
7.2.4 Right click on program **DB2REGI.cbl** and select **Open With** > **z Systems LPEX editor**



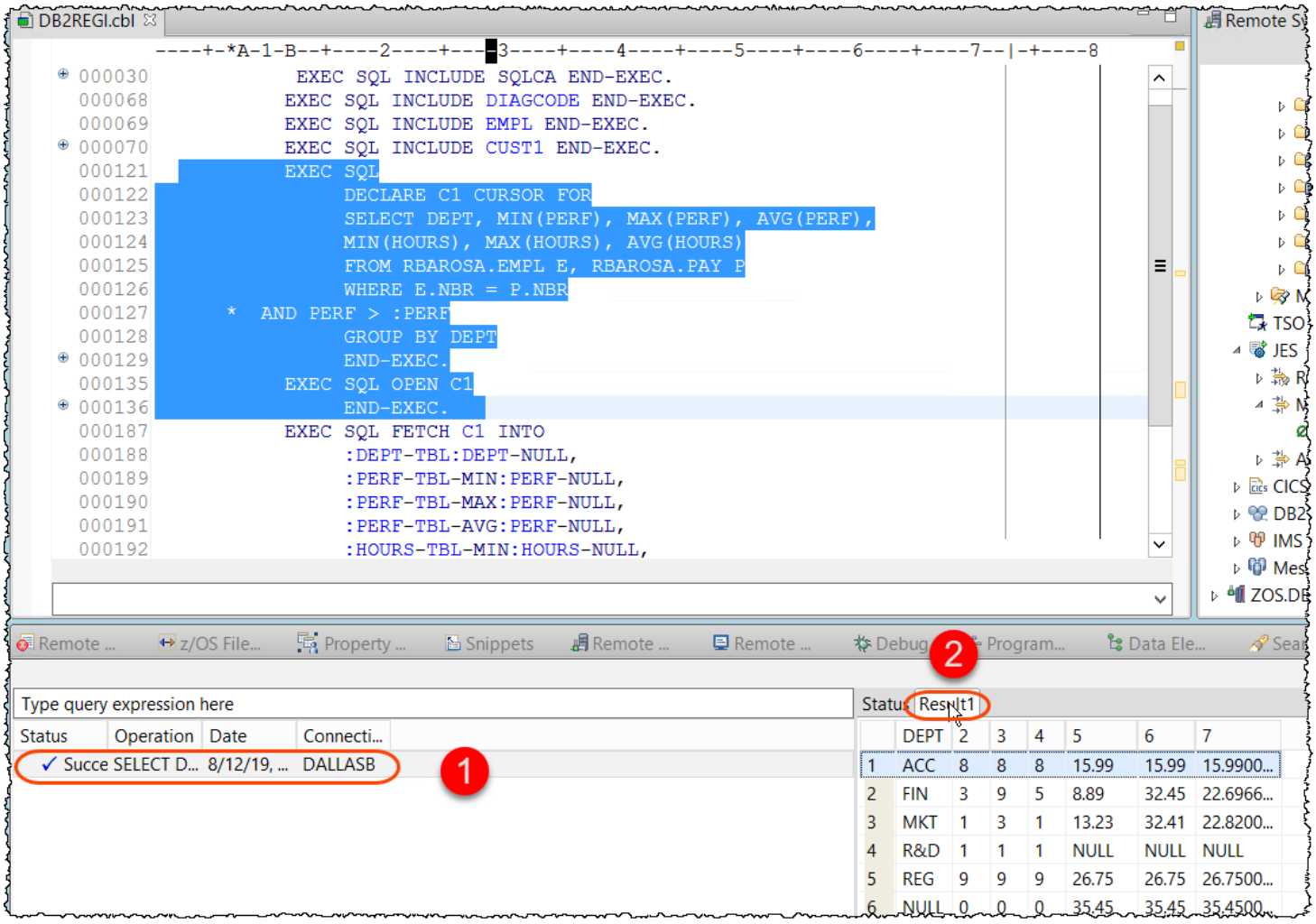
7.2.5 Right click and select **Filter View > Embedded SQL/CICS/DLI**


DB2REGI.cbl
Remote
Save Ctrl+S
Cut Ctrl+X
Copy Ctrl+Insert
Paste Shift+Insert
Select
Selected
Deselect Alt+U
Filter view
Show all Ctrl+W
Source
Preprocessor Statements
Refactor
Refresh SQL in Outline View
Occurrences in Compilation Unit Ctrl+Shift+R
Open Declaration F3
Divisions Ctrl+G
Code
Comments
Outline
Embedded SQL/CICS/DLI
Comment task tags
Copy Statements
Call Statements
DATA BY DEPT,
IMUM
/2015

7.2.6 **Highlight** the SQL statements as shown **right click** and choose **Run SQL**

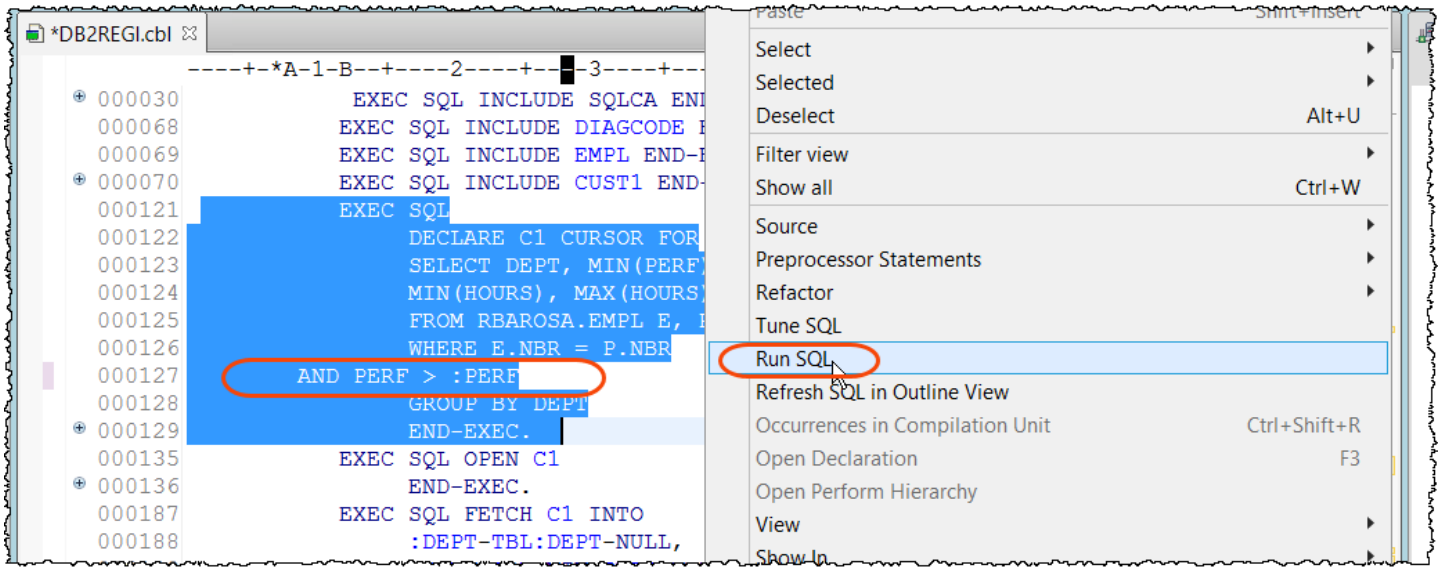


7.2.7 Click as below and you will have the query results

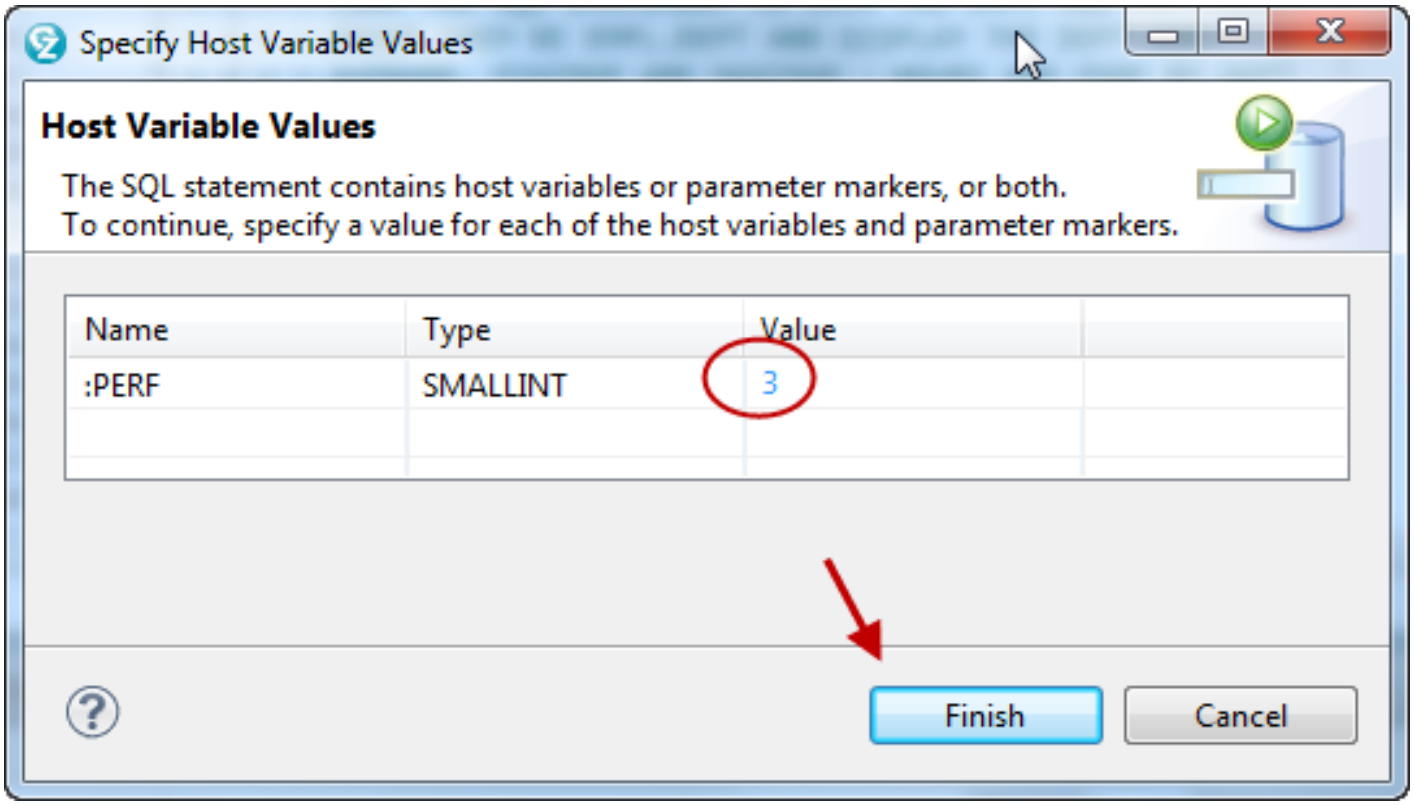

DB2REGI.cbl
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC SQL INCLUDE DIAGCODE END-EXEC.
EXEC SQL INCLUDE EMPL END-EXEC.
EXEC SQL INCLUDE CUST1 END-EXEC.
EXEC SQL
DECLARE C1 CURSOR FOR
SELECT DEPT, MIN(PERF), MAX(PERF), AVG(PERF),
MIN(HOURS), MAX(HOURS), AVG(HOURS)
FROM RBAROSA.EMPL E, RBAROSA.PAY P
WHERE E.NBR = P.NBR
AND PERF > :PERF
GROUP BY DEPT
END-EXEC.
EXEC SQL OPEN C1
END-EXEC.
EXEC SQL FETCH C1 INTO
:DEPT-TBL:DEPT-NULL,
:PERF-TBL-MIN:PERF-NULL,
:PERF-TBL-MAX:PERF-NULL,
:PERF-TBL-AVG:PERF-NULL,
:HOURS-TBL-MIN:HOURS-NULL,
Remote S
TSO
JES
CICS
DB2
IMS
ZOS.DE
Remote ...
z/OS File...
Property ...
Snippets
Remote ...
Remote ...
Debug
Program...
Data Ele...
Type query expression here
Status
Operation
Date
Connecti...
Succe SELECT D...
8/12/19, ...
DALLASB
1
2
Status
Result1
DEPT
2
3
4
5
6
7
1 ACC 8 8 8 15.99 15.99 15.9900...
2 FIN 3 9 5 8.89 32.45 22.6966...
3 MKT 1 3 1 13.23 32.41 22.8200...
4 R&D 1 1 1 NULL NULL NULL
5 REG 9 9 9 26.75 26.75 26.7500...
6 NULL 0 0 0 35.45 35.45 35.4500...

7.2.8 Modifying the SQL query..

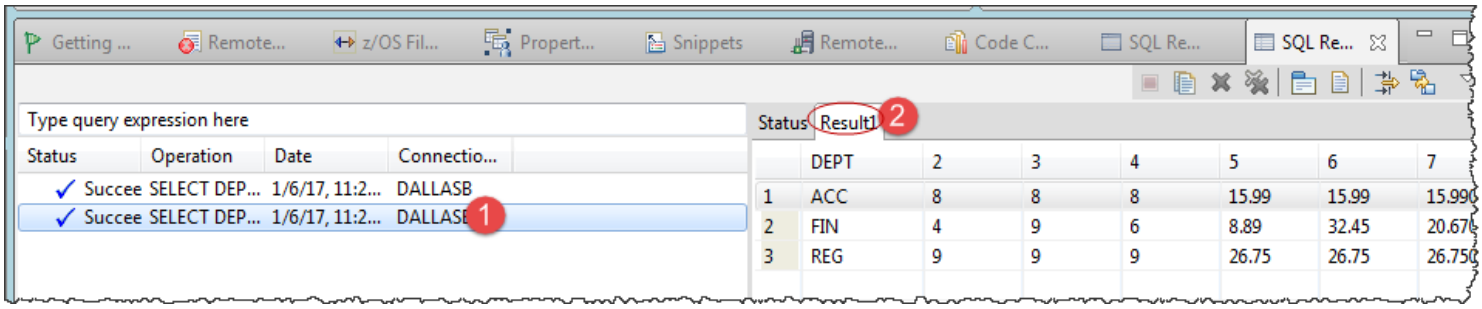
Remove the COBOL comment (***)** from the line **127**,
Select the SQL statements again right click and choose **Run SQL**



7.2.9 Specify a value like **3** and click **Finish**

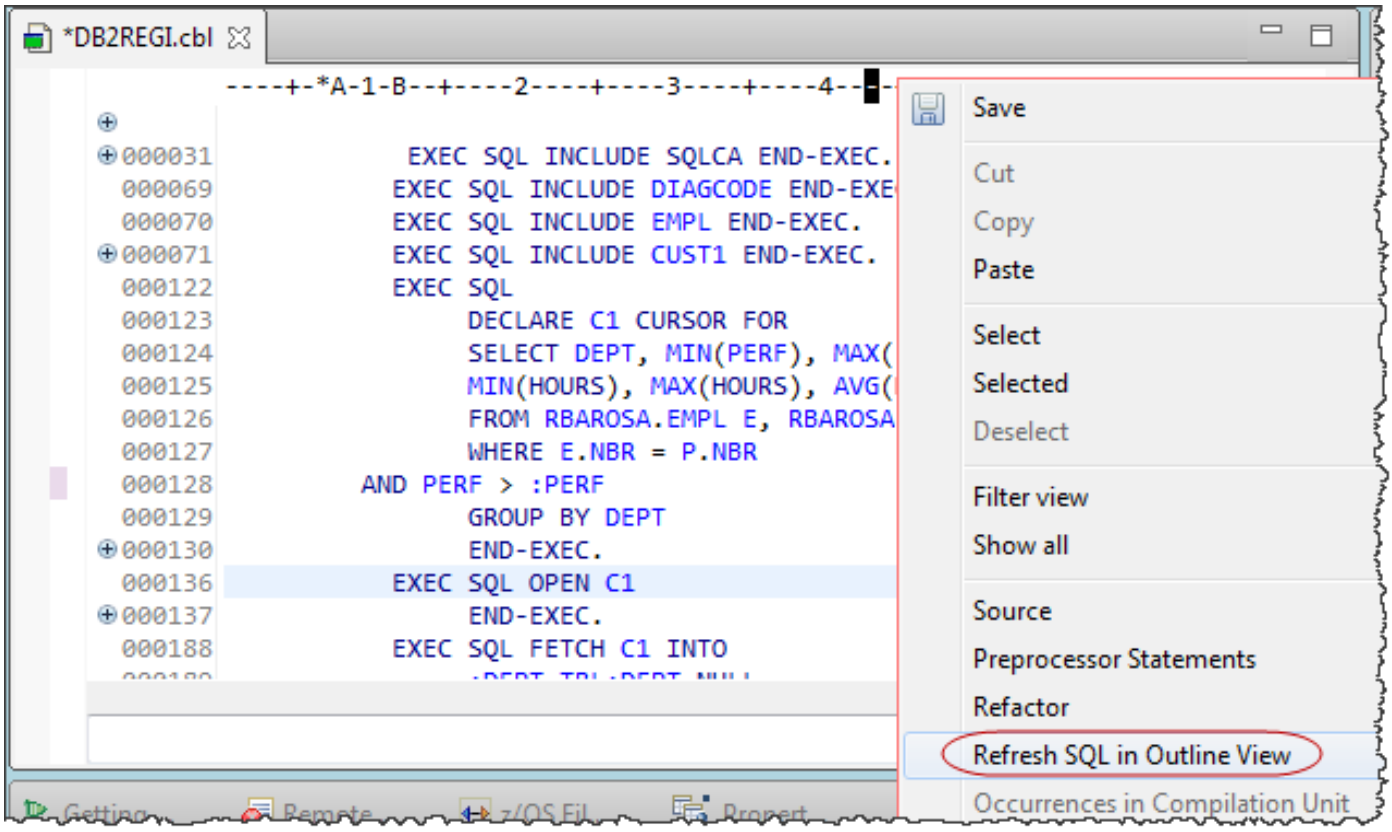


7.2.10 The new results now will be:

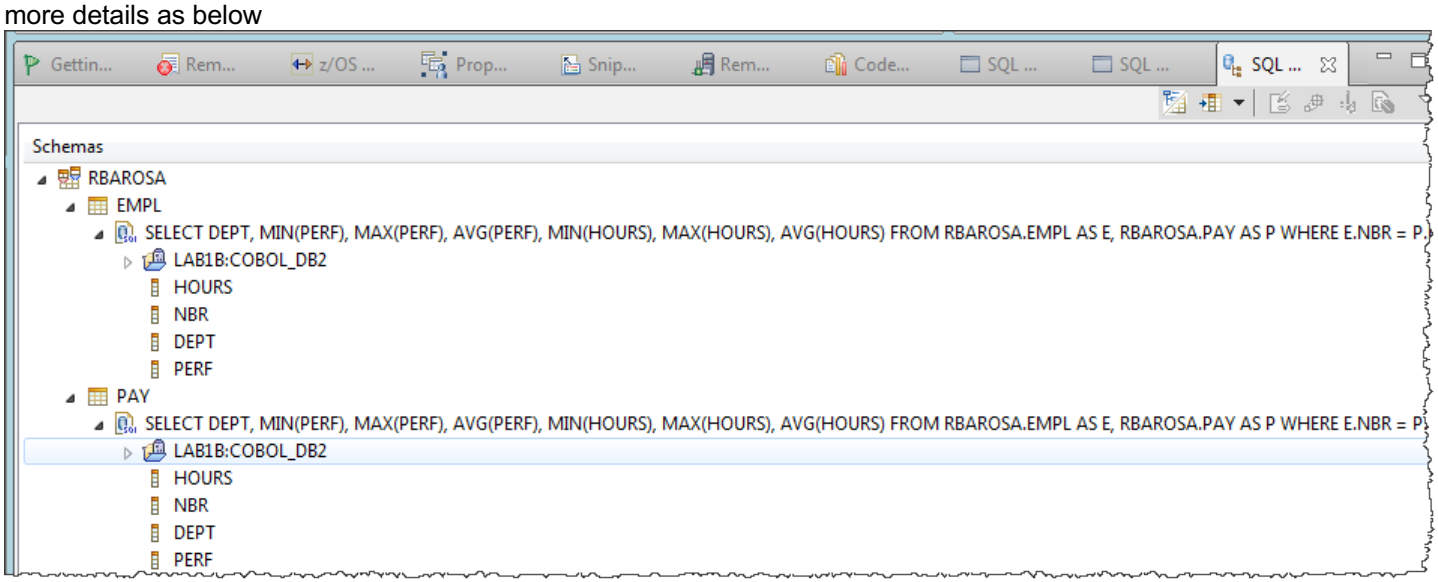

Getting ...
Remote...
z/OS Fil...
Propert...
Snippets
Remote...
Code C...
SQL Re...
SQL Re...
Type query expression here
Status
Operation
Date
Connectio...
Succee SELECT DEP... 1/6/17, 11:2... DALLASB
Succee SELECT DEP... 1/6/17, 11:2... DALLASB
1
Status
Result1
2
DEPT
2
3
4
5
6
7
1 ACC 8 8 8 15.99 15.99 15.990
2 FIN 4 9 6 8.89 32.45 20.670
3 REG 9 9 9 26.75 26.75 26.750

___ 7.3 Using the SQL Outline view

7.3.1 Without selecting any statement, **right click** and select **Refresh SQL in Outline View**

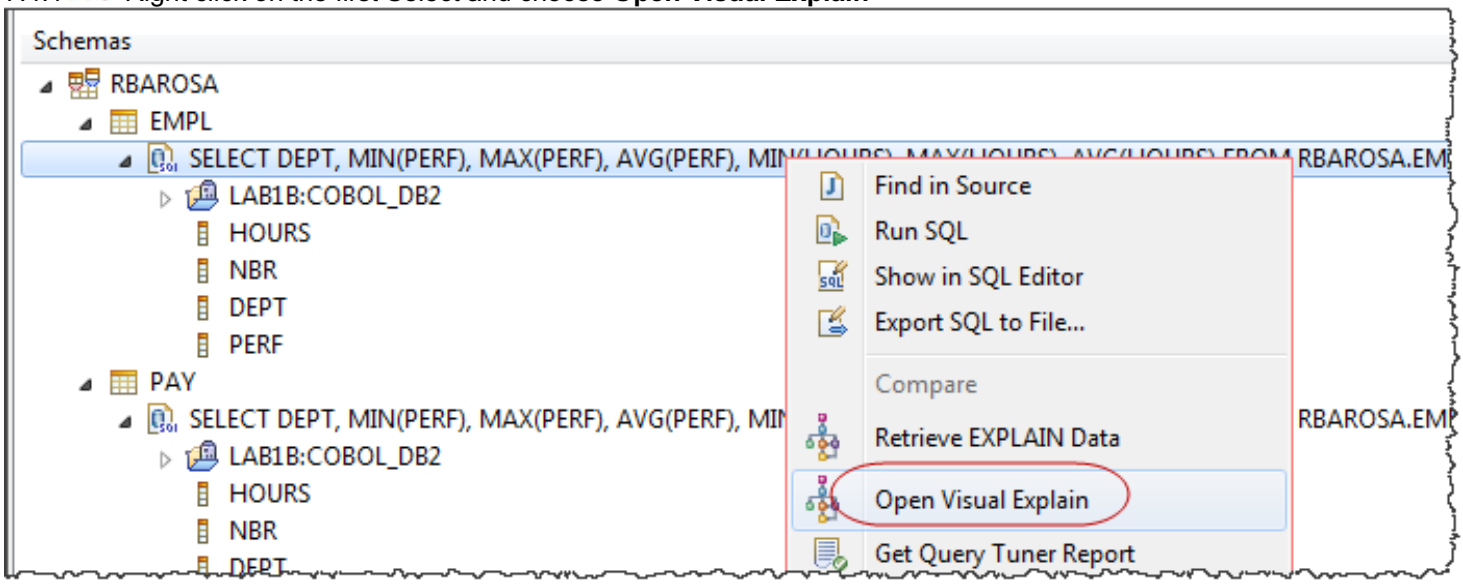


7.3.2 Using *SQL Outline View*, expand **RBAROSA**, **EMPL, PAY** and **the SQL statements** and you will have more details as below

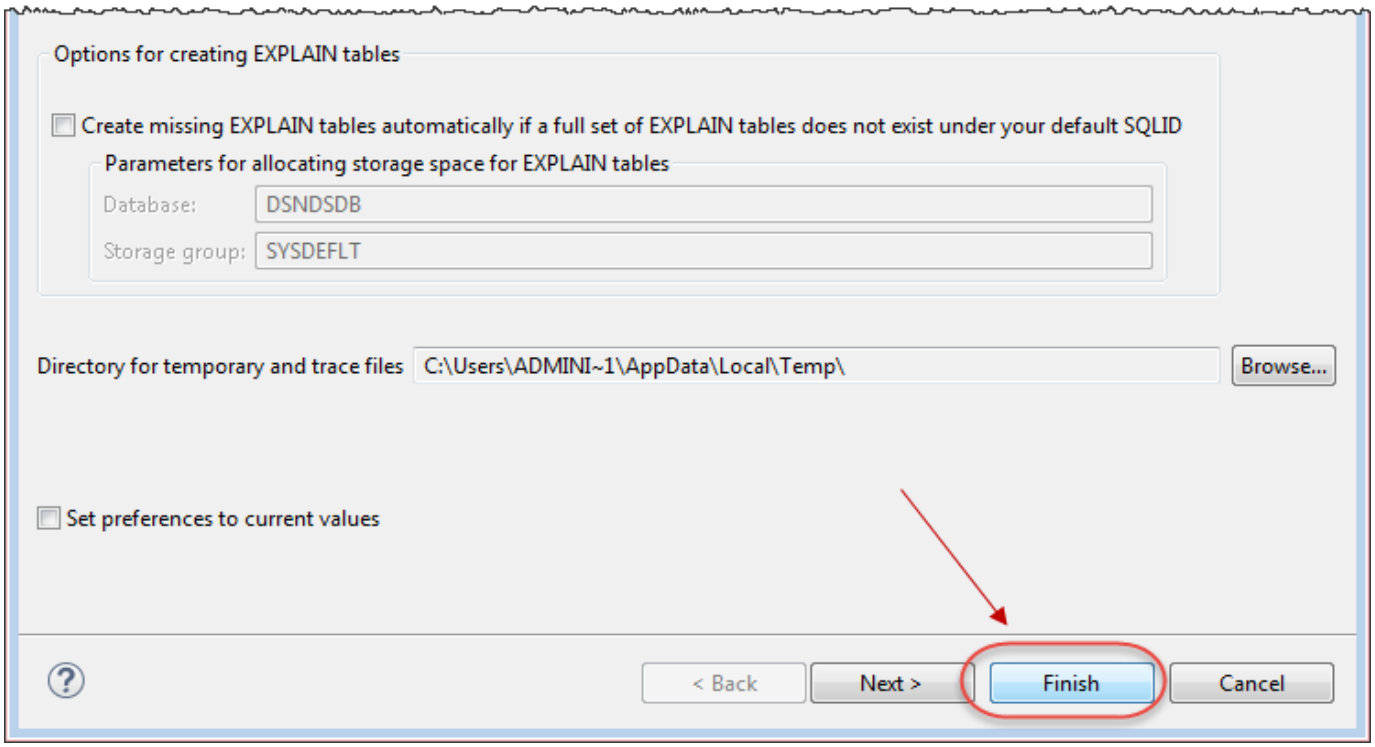


___ 7.4 Using SQL Visual Explain

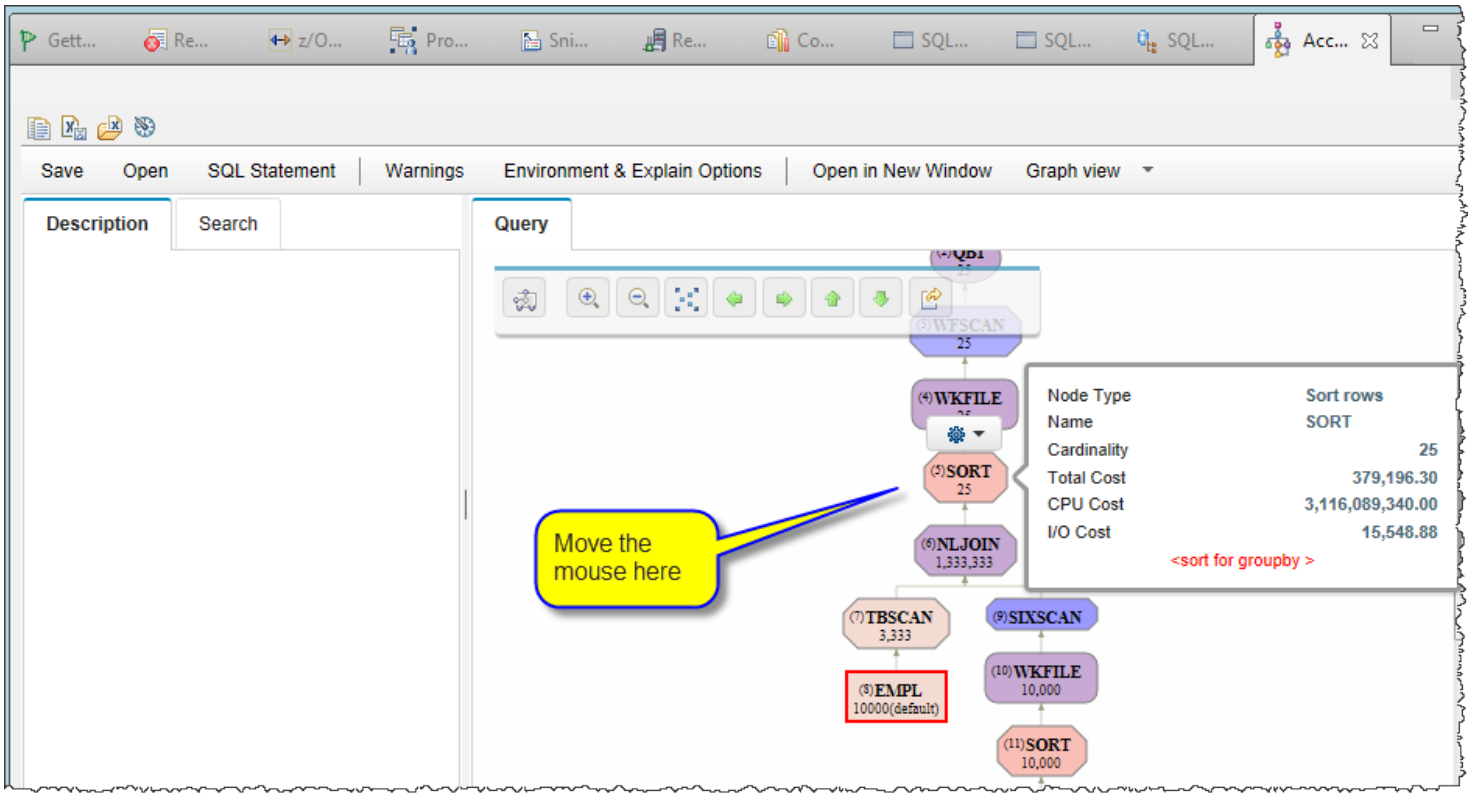
7.4.1 Right click on the first Select and choose **Open Visual Explain**



7.4.2 Click **Finish**. A new report view (*Access Plan Diagram*) will be generated in the bottom of the screen...

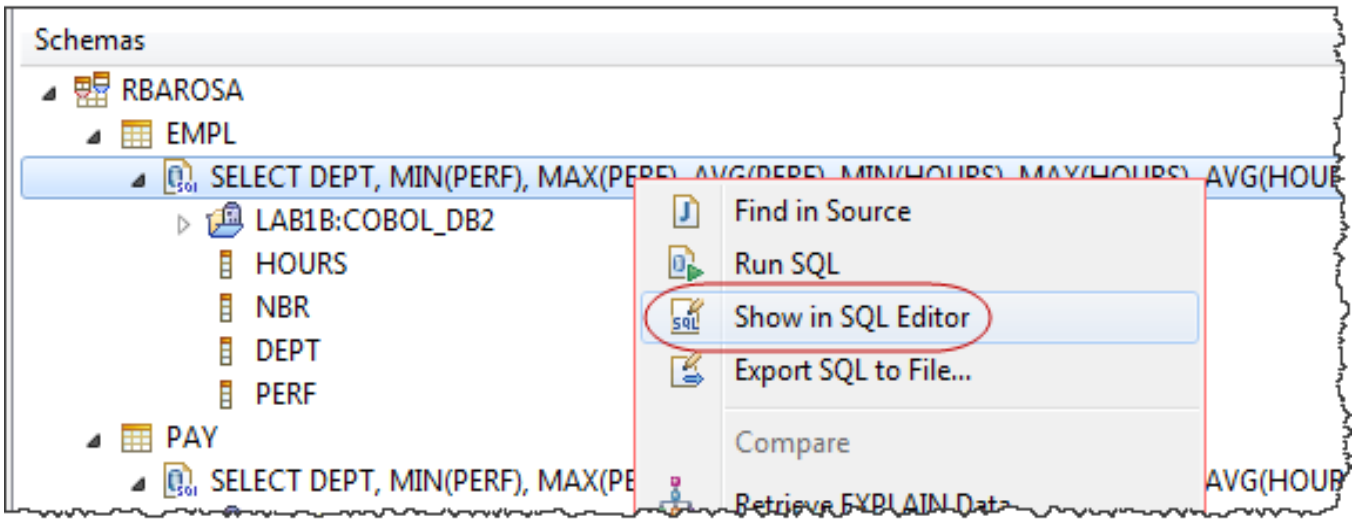


7.4.3 Double click on the tab to have a bigger view.
You will have the report below to work with the SQL explain

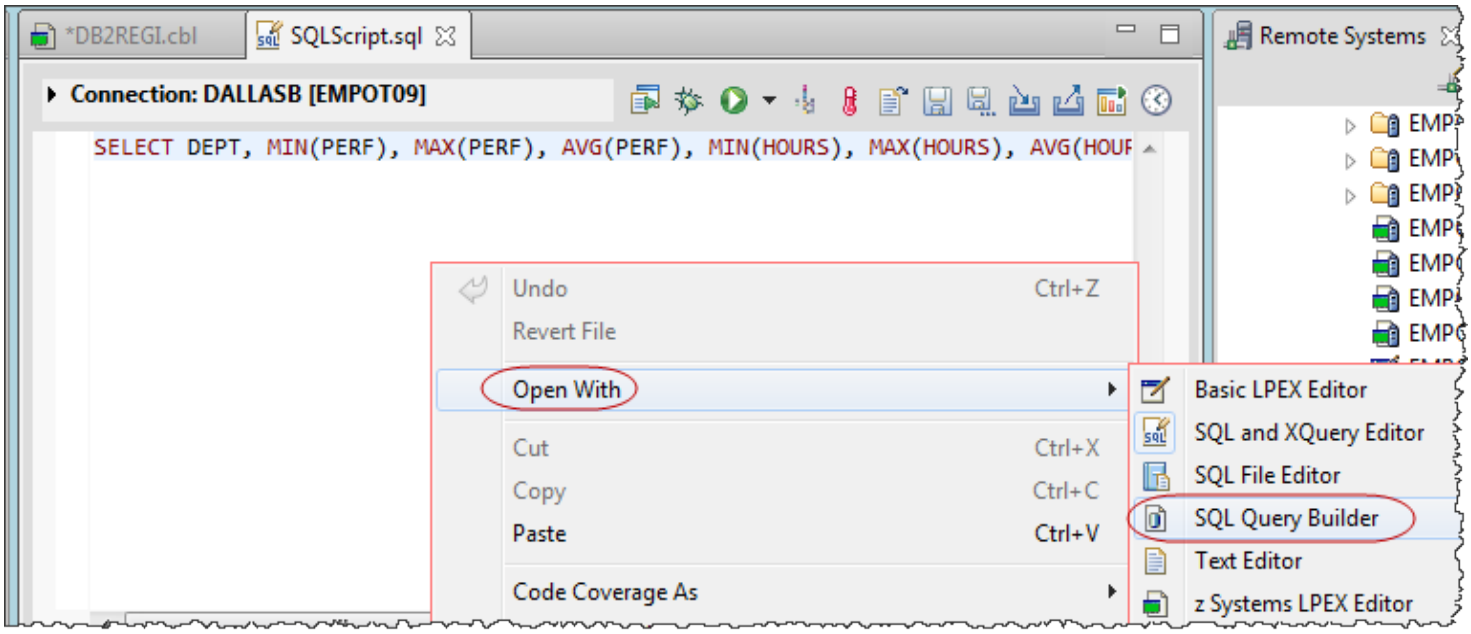


___ 7.5 Reverse engineer the Query

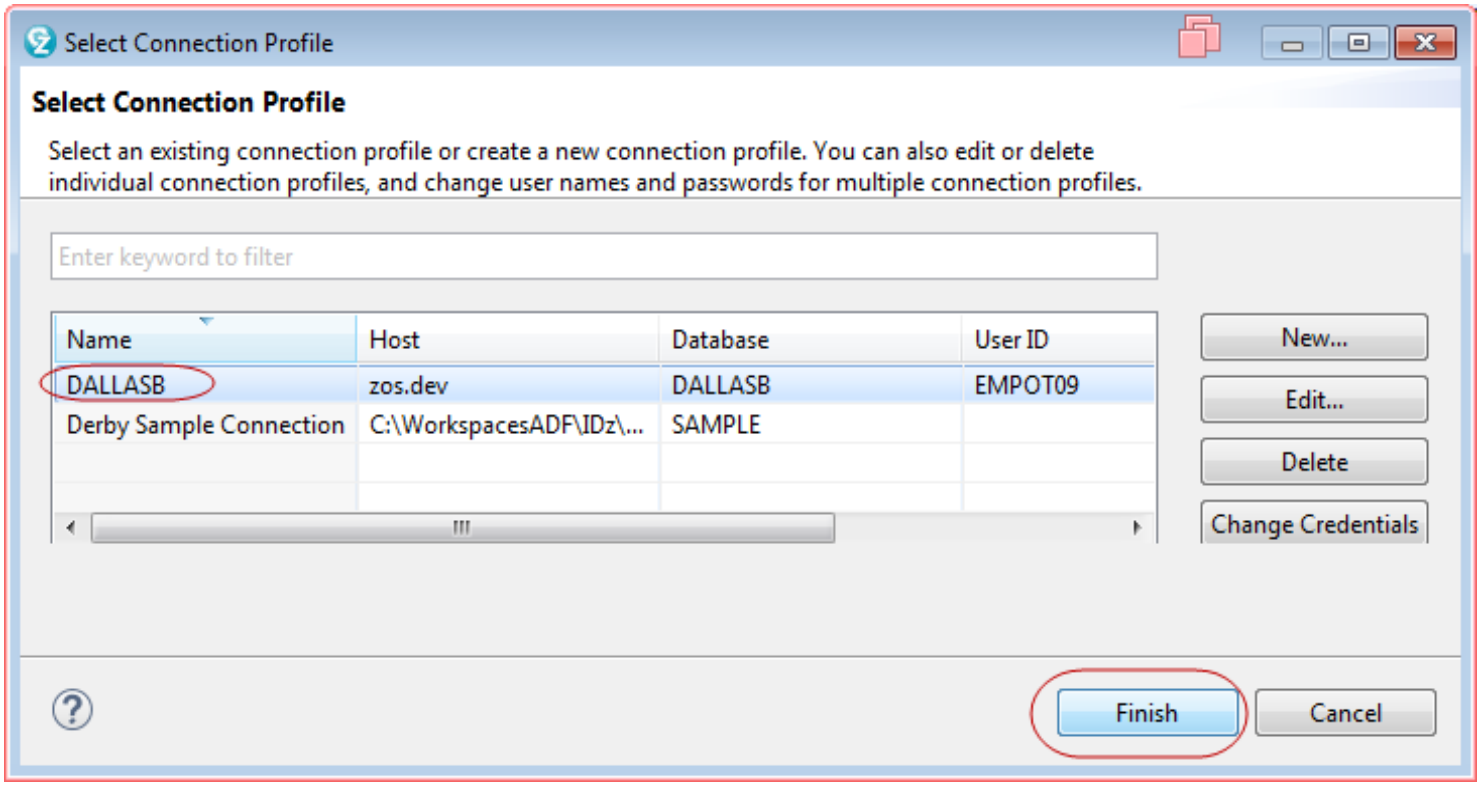
7.5.1 ⏭ Using the *SQL Outline* view, **right click on the Select** and choose **Show in SQL Editor**



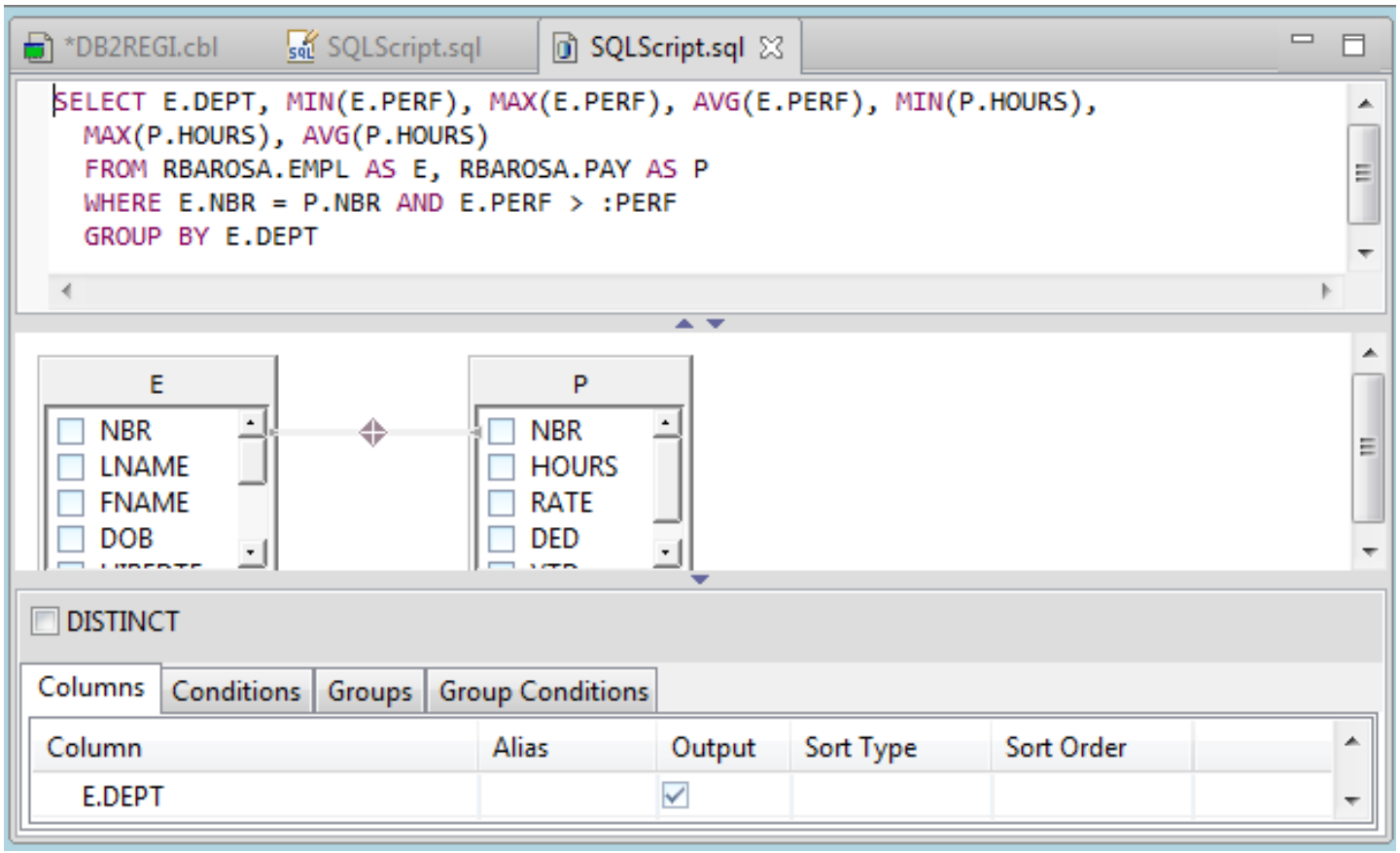
7.5.2 When opened,. select **Open With > SQL Query Builder**



7.5.3 Choose **DALLASB** and click **Finish**



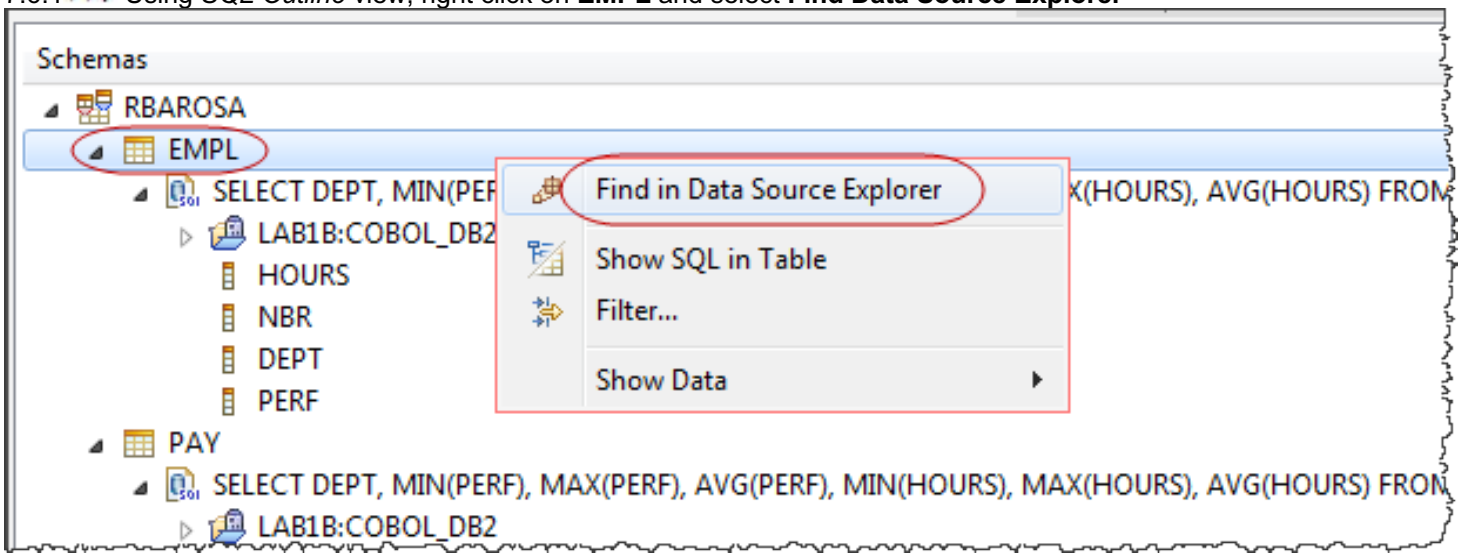
7.5.4 Now we have the graphical editor



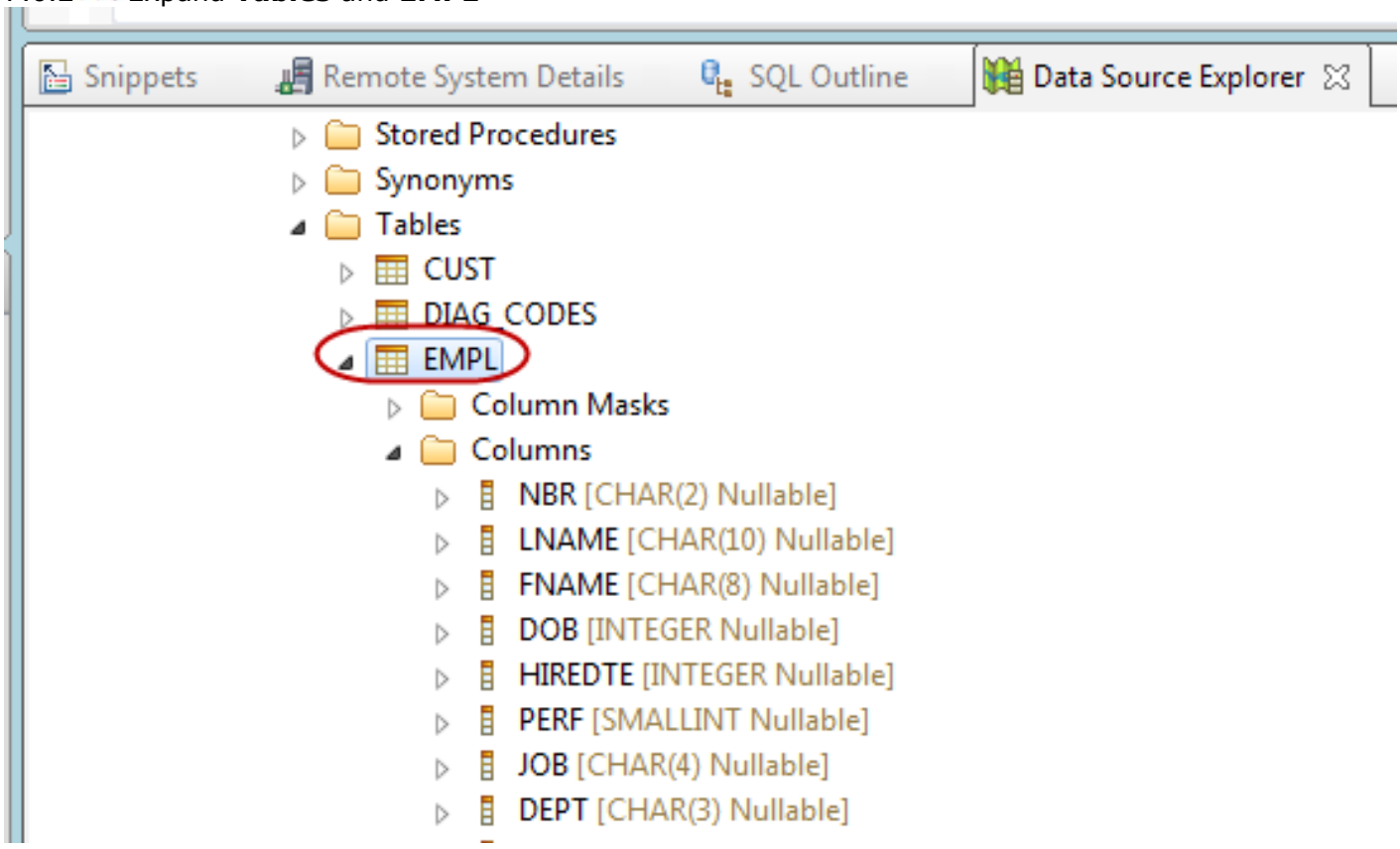
7.5.5 ⏭ You might use this graphical editor to make changes on SQL statements.
Notice that the changes reflects on the COBOL code..

___ 7.6 Displaying DB2 table content

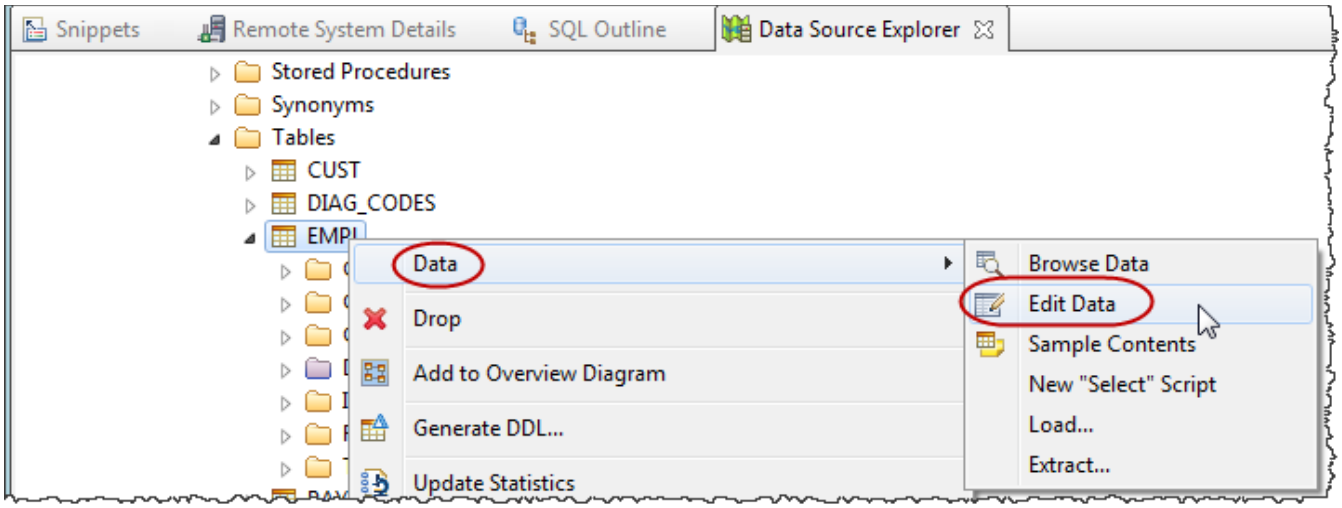
7.6.1 Using *SQL Outline* view, right click on **EMPL** and select **Find Data Source Explorer**



7.6.2 Expand **Tables** and **EMPL**

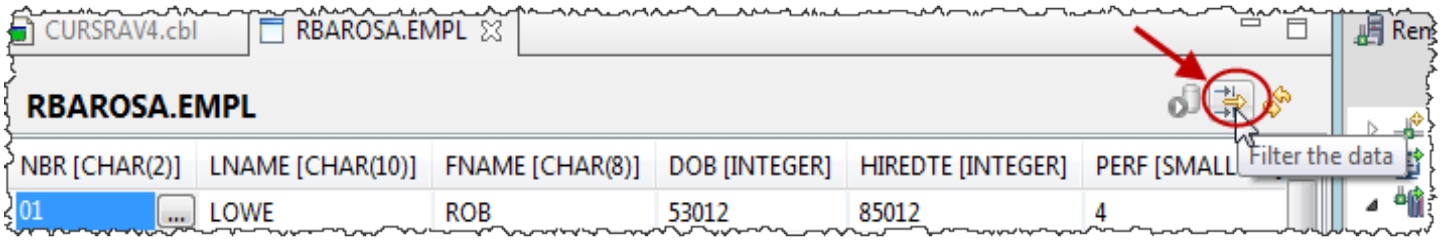


7.6.3 Right click on **EMPL** and select **Data > Edit Data**

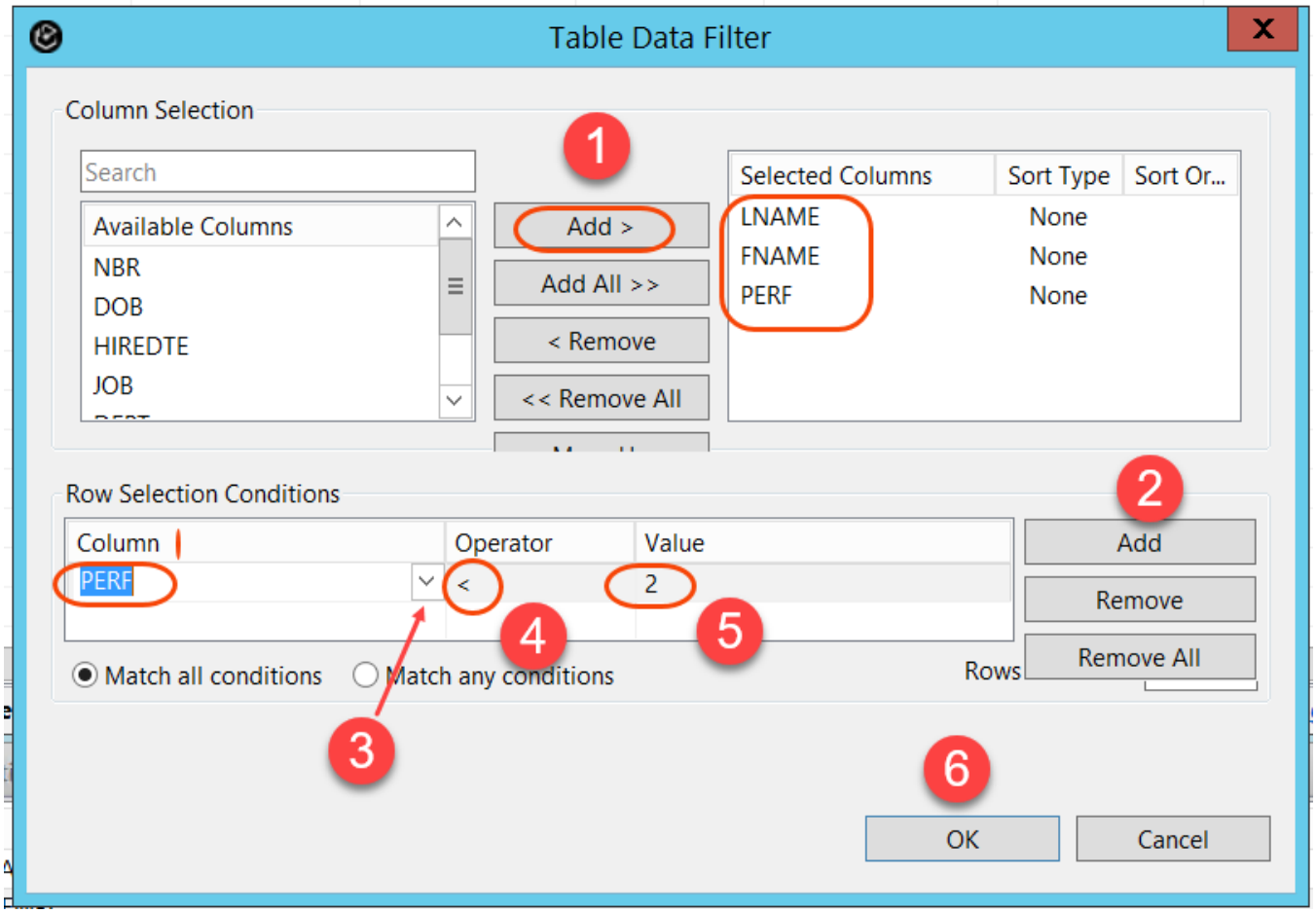


Filtering data results.

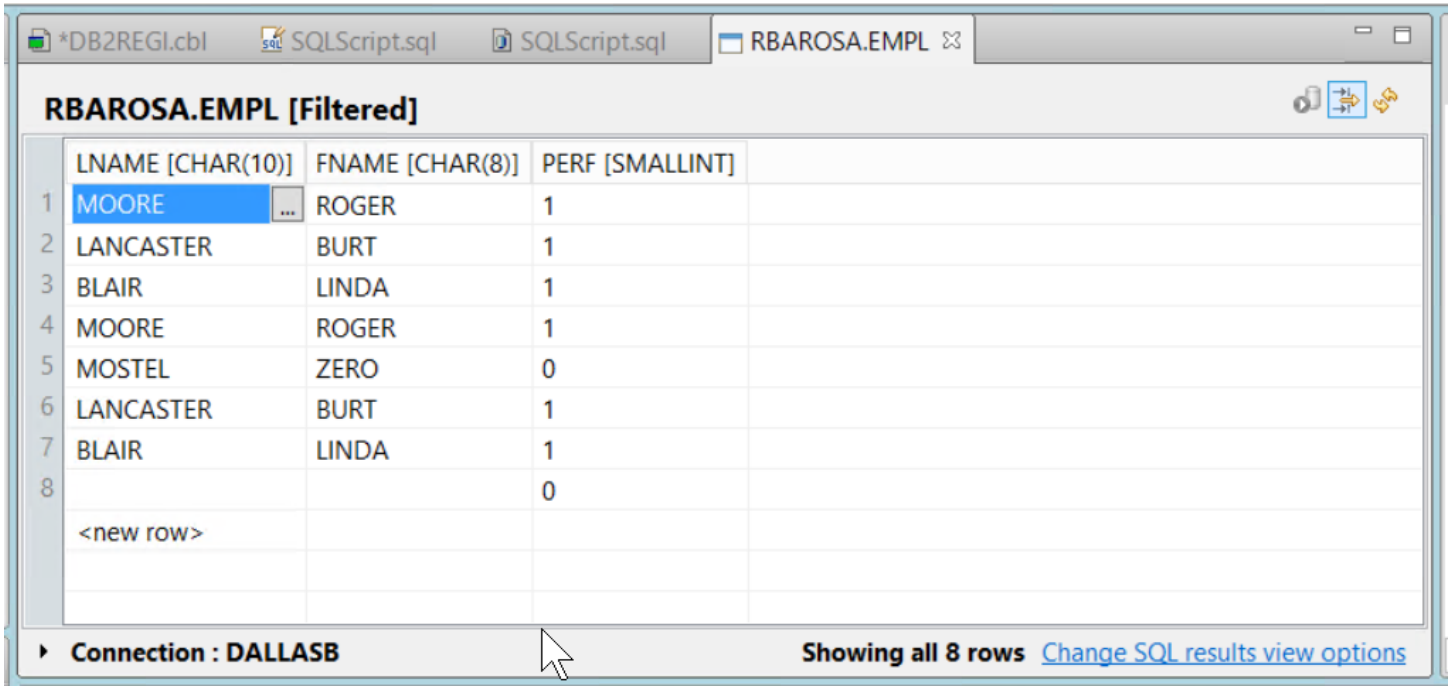
7.6.4 Click **Filter** the data icon



7.6.5 On the *Table Data Filter* dialog, using **Add >** button, select **LNAME, FNAME** and **PERF.**
Using the second **Add** button, use the pull down and select **EMPL.PERF** for *Column*, **<** as *Operator* and type **2** as *Value*.



7.6.6 Clicking **OK** you will get the results below:



7.6.7 Close all editors pressing **Ctrl + Shift + F4.** Or just click on the . of each opened editor

Say **NO** to saving the COBOL program.

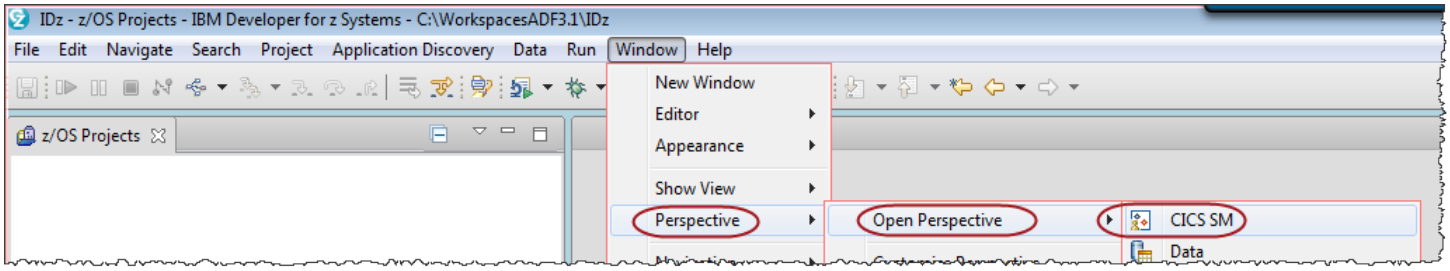
Section 8. (Optional) Using File Manager

ADFz may optionally have the File Manager (FM) plugin installed. FM works with a broad spectrum of z/OS files and data bases, including VSAM, IAM, and QSAM files, PDS and libraries, DB2 and IMS databases, HFS files, OAM files, CICS queues, MQ queues, and tapes.
File Manager provides powerful formatted editors and viewers, and also provides a full complement of on-line and batch utilities to copy, extract, and load data, to create files and databases, compare and print, and many other utility functions.
File Manager has a conventional 3270 interface that can be accessed from TSO or CICS. here is also an eclipse GUI interface that is available on ADFz.
File Manager has many capabilities we will explore just few examples on this optional lab.
Note that this exercise could be also done with either IDz version 14 or version 15. Here we are using version 14, but instructions will be the same.

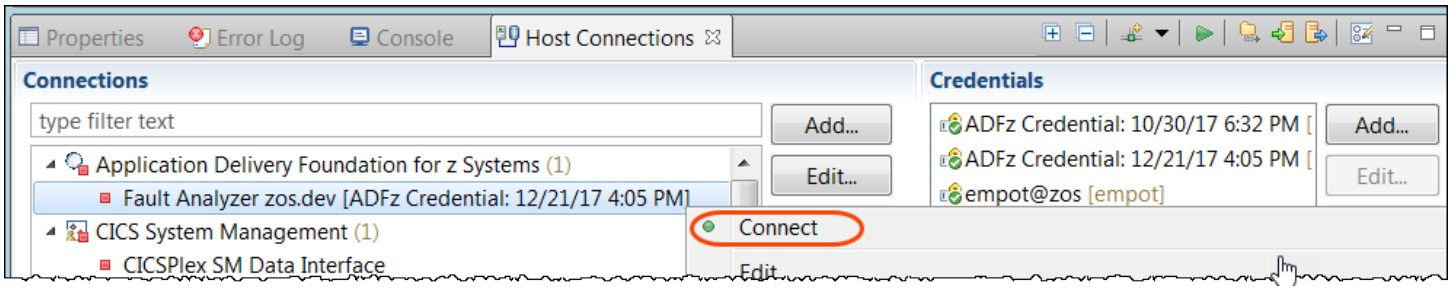
___ 8.1 Verify that you are connected to the PDTOOLS Common components

On step, 3.1 you have used the PDTOOLS Common Components. Let's be sure that you are still connected.

8.1.1 Go to **CICS SM** perspective using
Window > Perspective> Open Perspective > CICS SM
(If CICS SM is not shown click **Other** and find **CICS SM**)



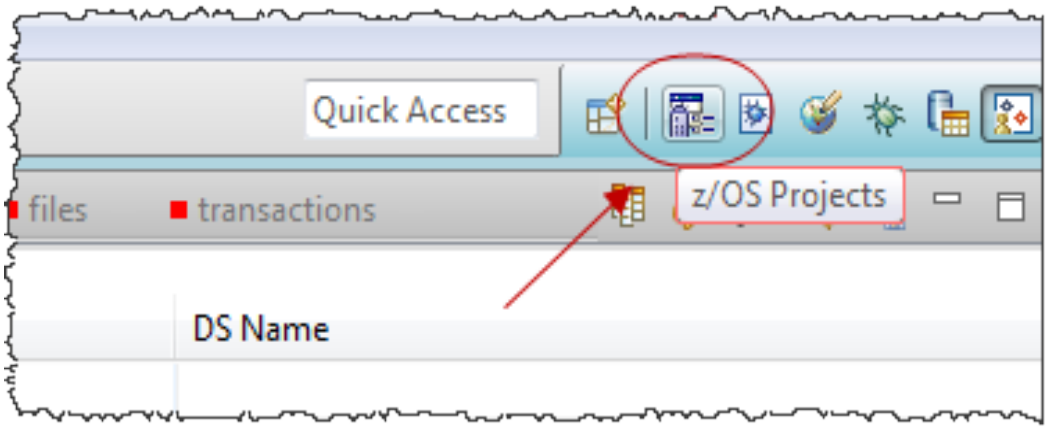
8.1.2 Click on **Host Connections** tab (bottom) and verify that **Application Delivery Foundation for z Systems** has a green icon: .
If it is not green you must connect it. **Right click and select Connect**



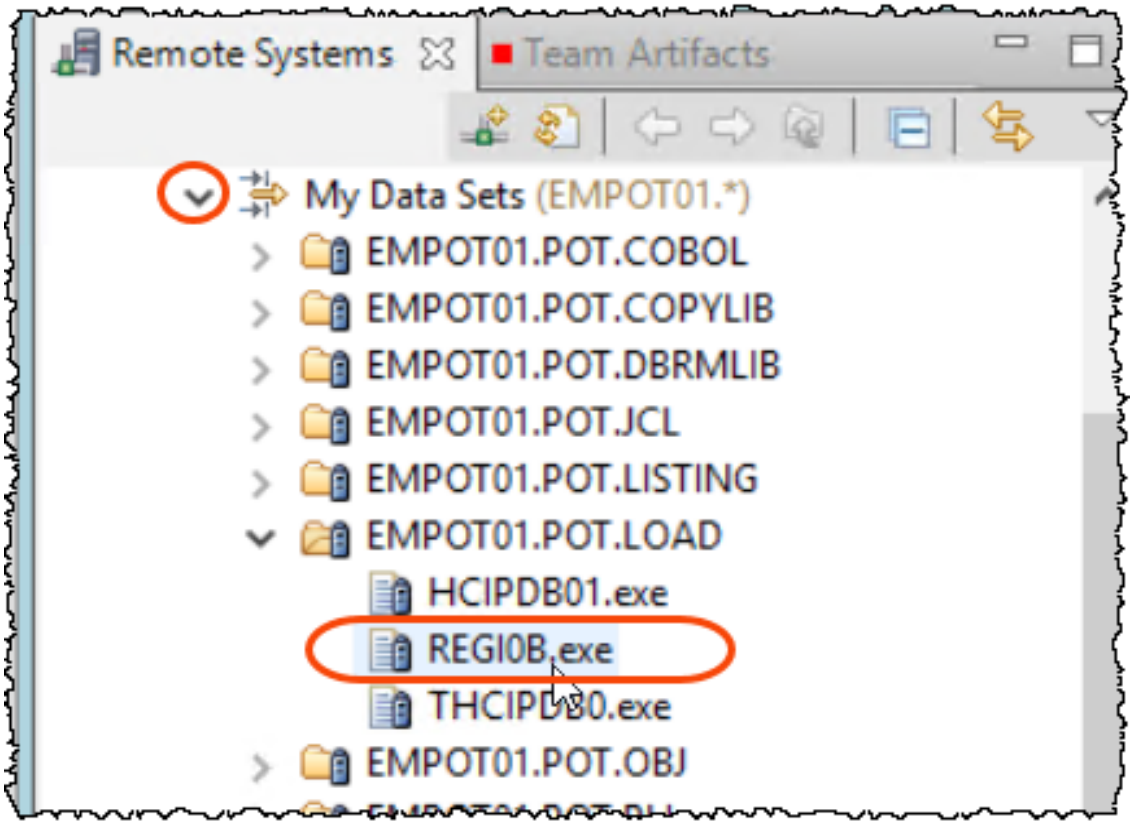
___ 8.2 Using View Load Module utility

You can use the **View Load Module** utility function to print a list of the symbols (CSECTs, common sections, entry points, and ZAPs) in a load module. You also could compare load modules, using the Compare wizard.
This page details the mapping of fields to batch parameters and miscellaneous notes. For the full description of each parameter, refer to the IBM File Manager Users Guide and Reference.

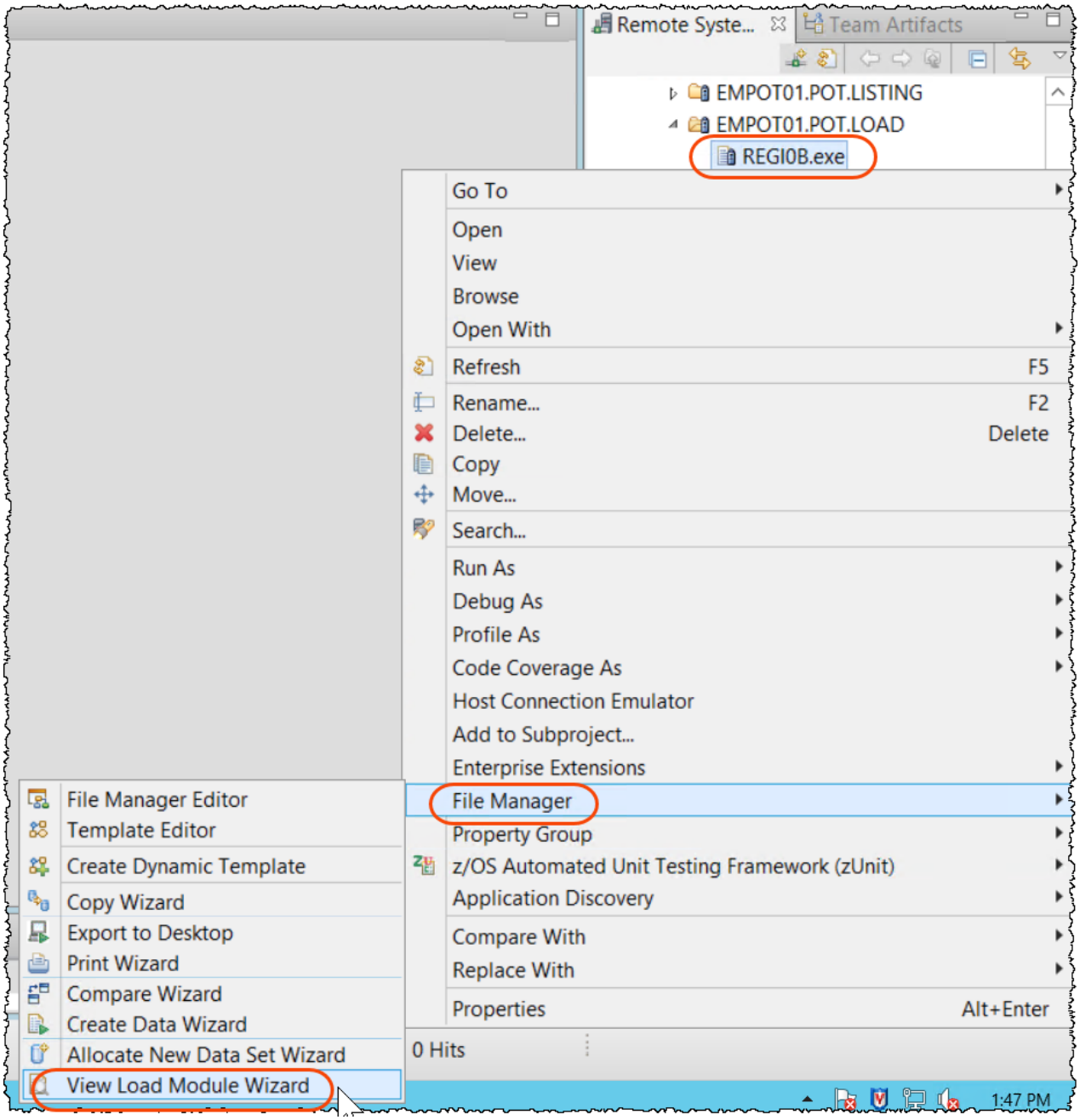
8.2.1 Go to **z/OS Projects** perspective clicking on the upper top right icon as below



8.2.2 Using Remote System view on the right, expand **MVS Files**, **My Data Sets** and **EMPOT01.POT.LOAD** and you should see the load module **REGI0B.exe** that you created before..

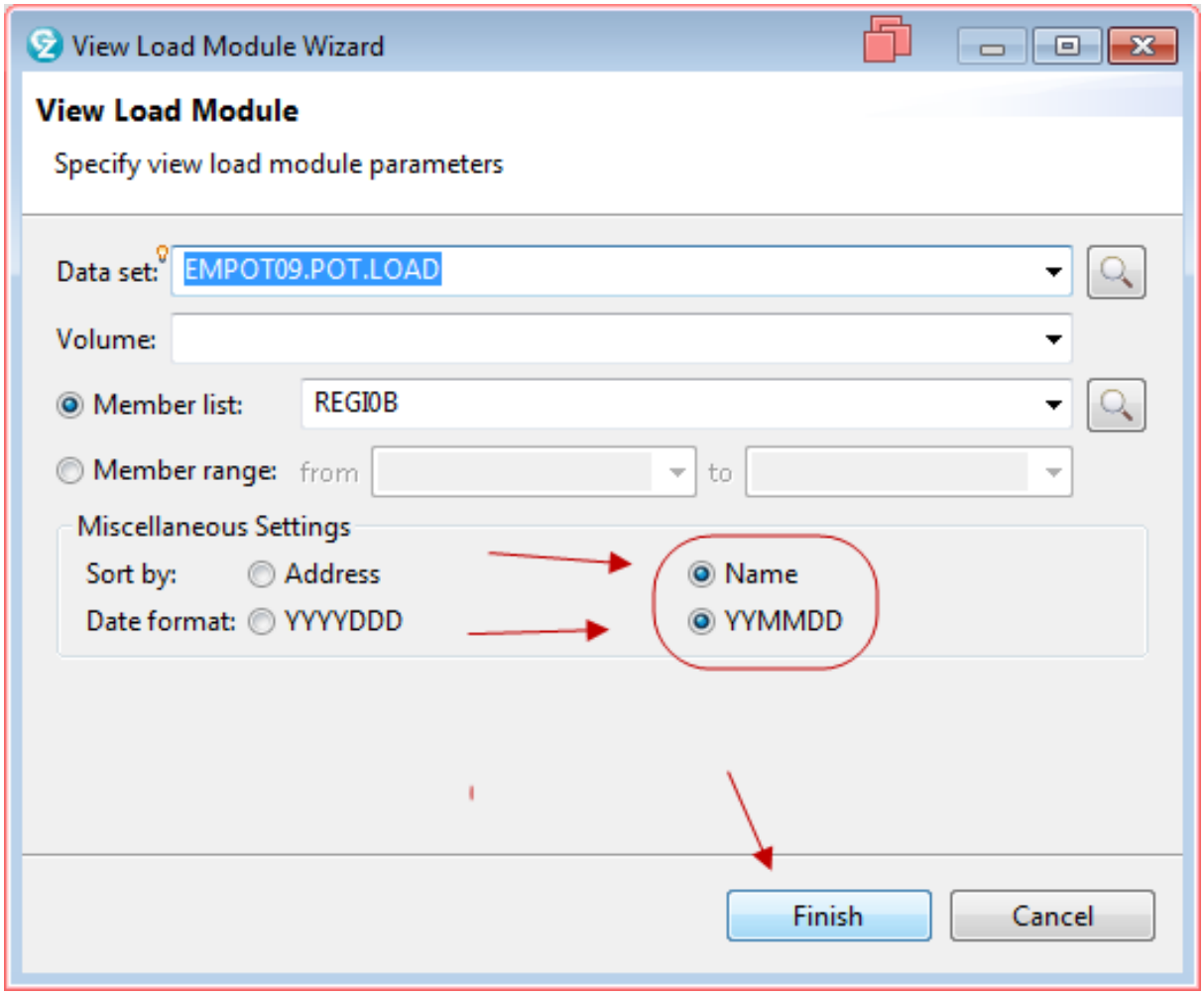


8.2.3 Right click on **REGI0B.exe** and select **File Manager** and **View Load Module Wizard**.



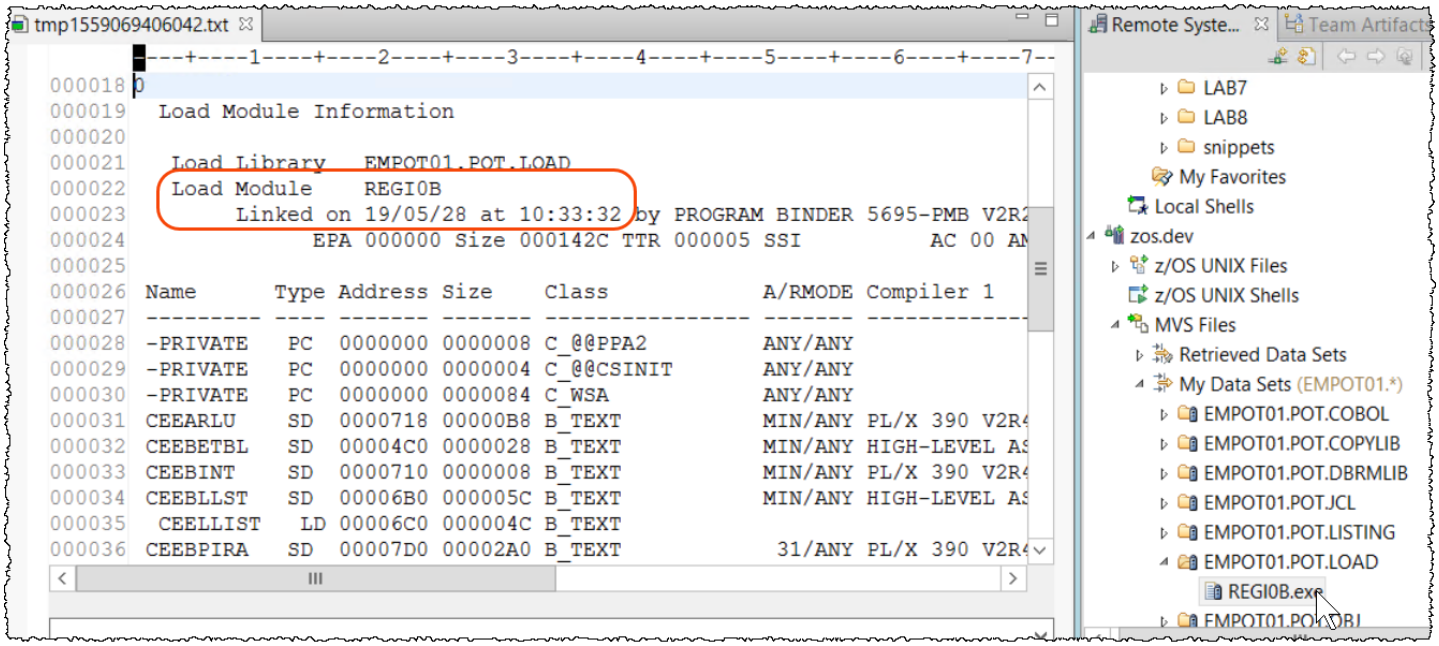
If you get a *Sign on* dialog for *Fault Analyzer* use **empot01** credentials and click **OK**

8.2.4 Select **Name** to sort by name and **YYMMDD** to show the date on this format and click **Finish**.



8.2.5 The result is below..

Scroll down and you may find interesting the date and time that this module was linked as well other information. You also could compare load modules.



Notice: The date that is shown is when your REGI0B module was created and will be different from what is shown on your report.

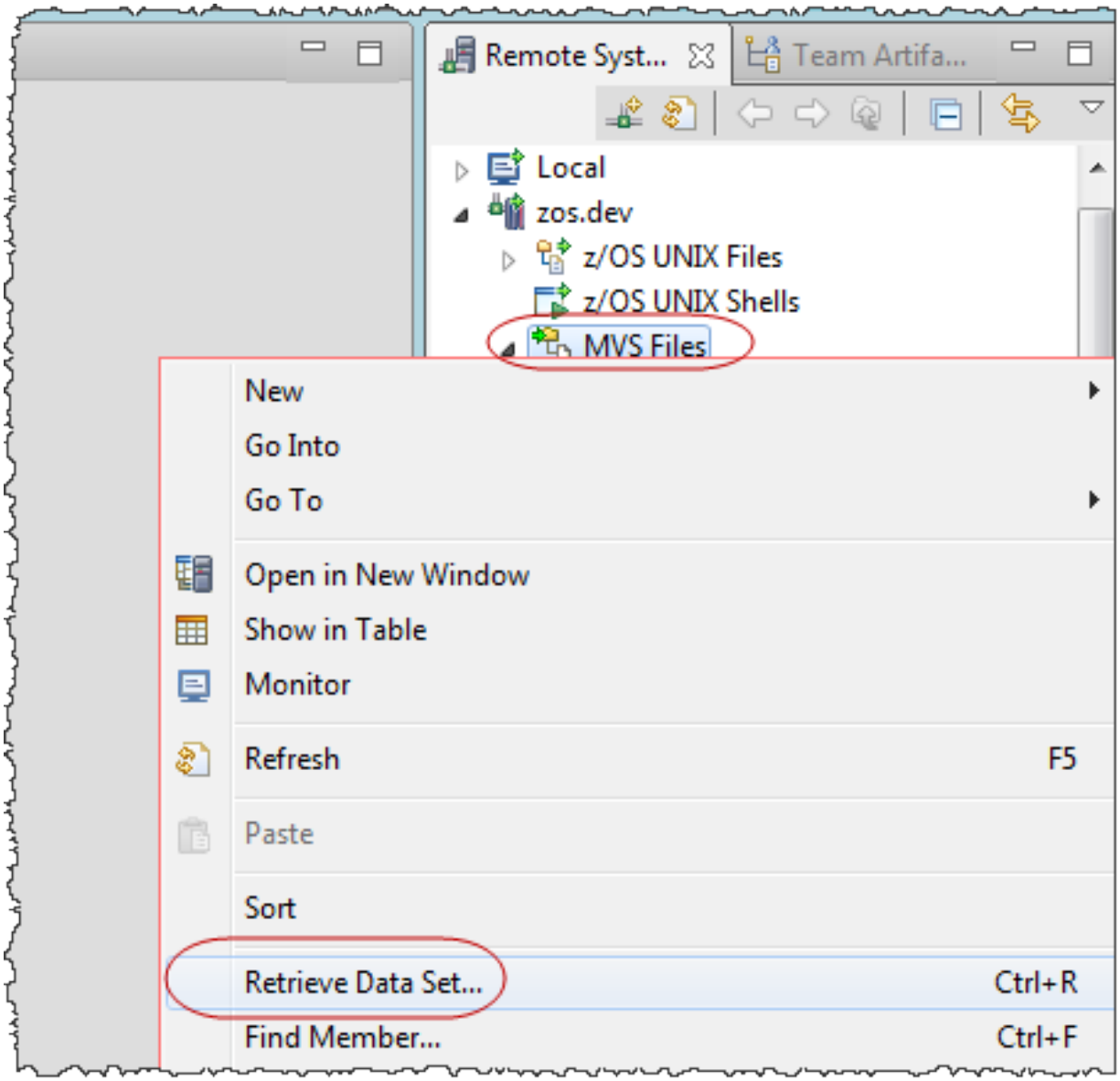
8.2.6 Close the edited file. Can use **CTRL + Shift + F4** to close all opened editors. Or just click on the . of each opened editor

___ 8.3 Working with z/OS data sets

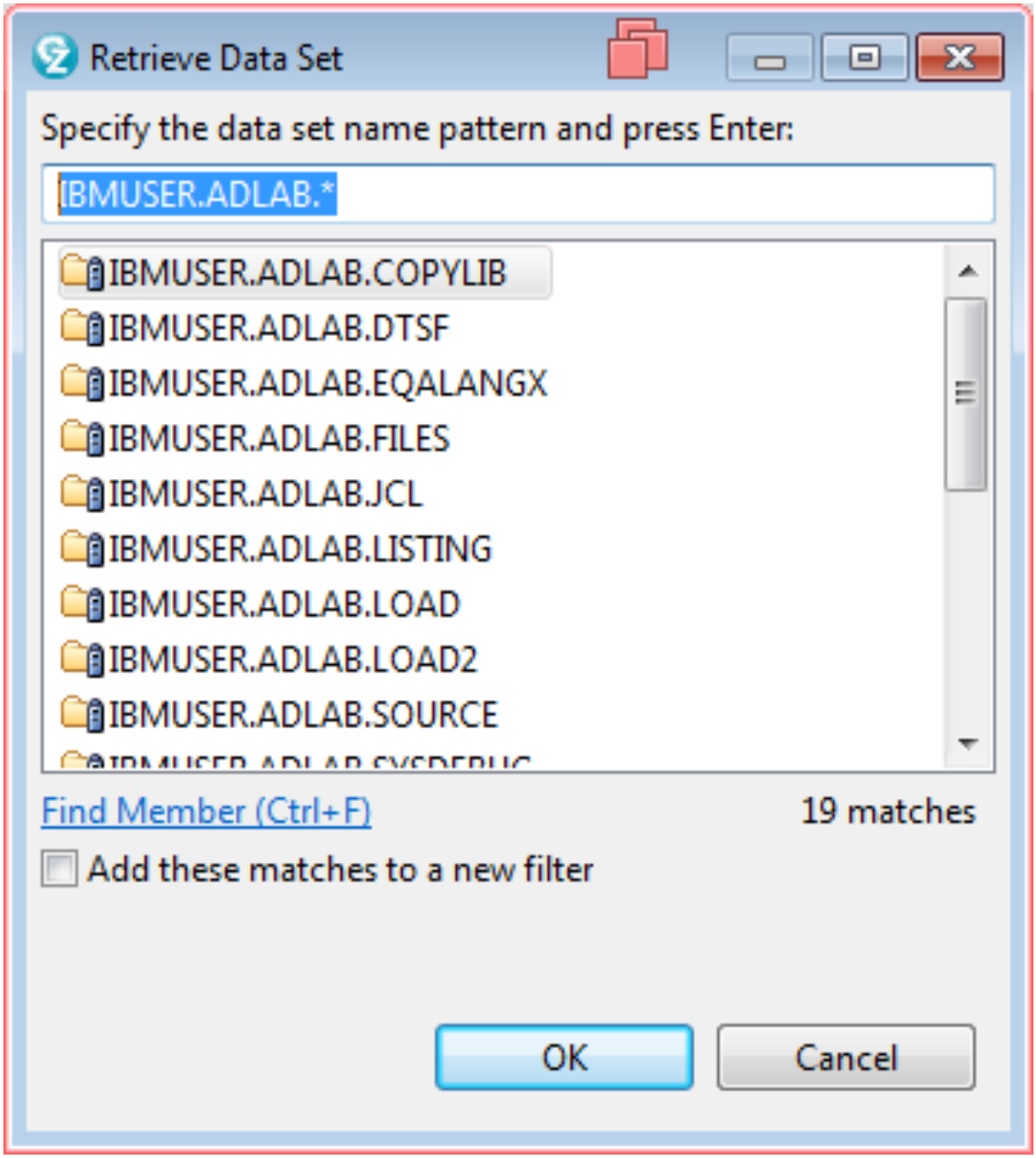
In this section, you will open a variable blocked sequential dataset in the editor or viewer, using a copybook as a layout.
You will use the very basic features of the editor, including navigating a file, we will not cover finding data in a file, changing data, inserting, deleting and sorting records.

8.3.1 Before you start mapping the dataset using a COBOL COPYBOOK, let us copy the copybook that maps the file to your PDS.

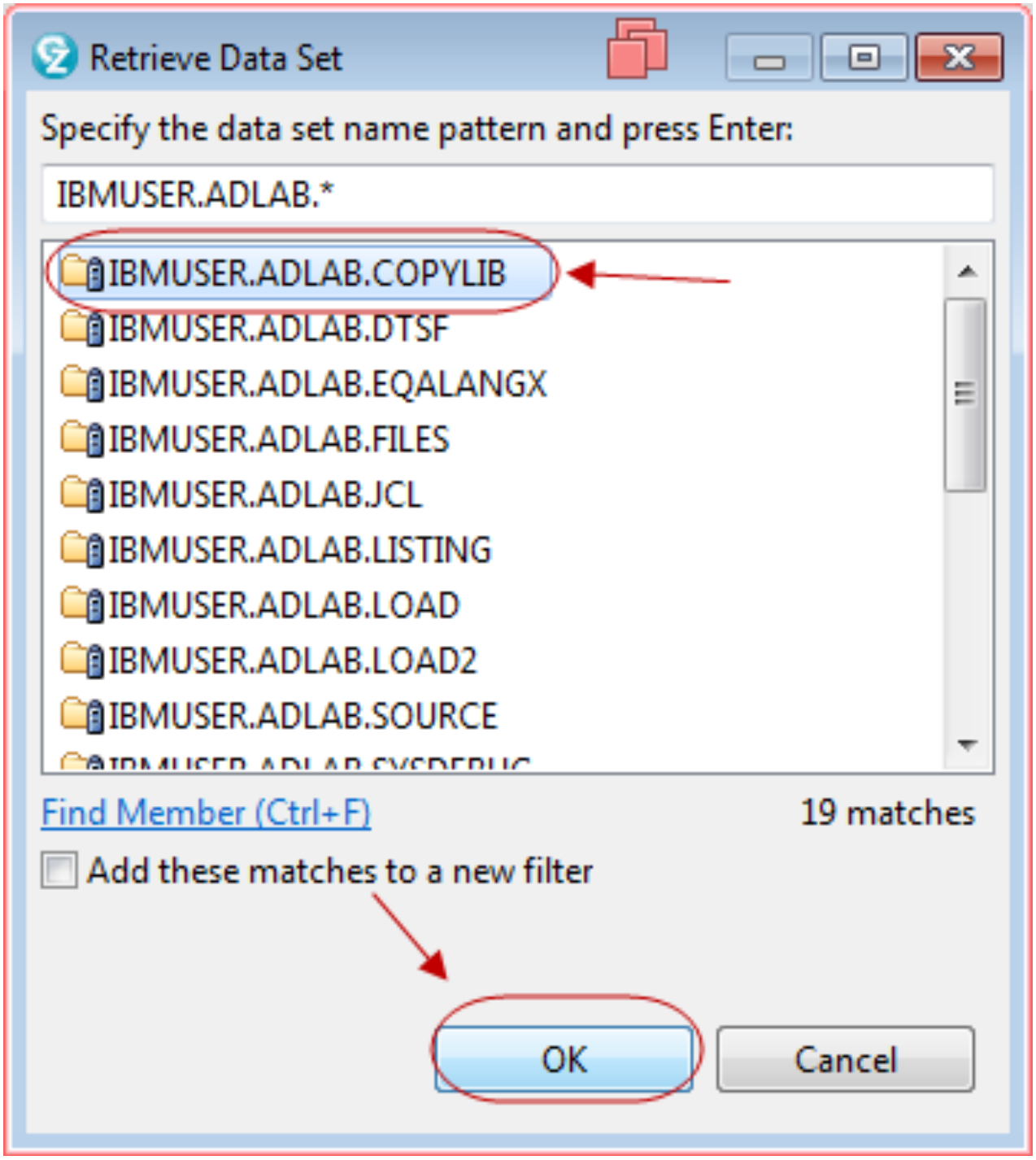
⏭ Using the *Remote Systems* view, right click on **MVS Files** and select **Retrieve Data Set** ….
This is equivalent at the TSO 3.4 function.



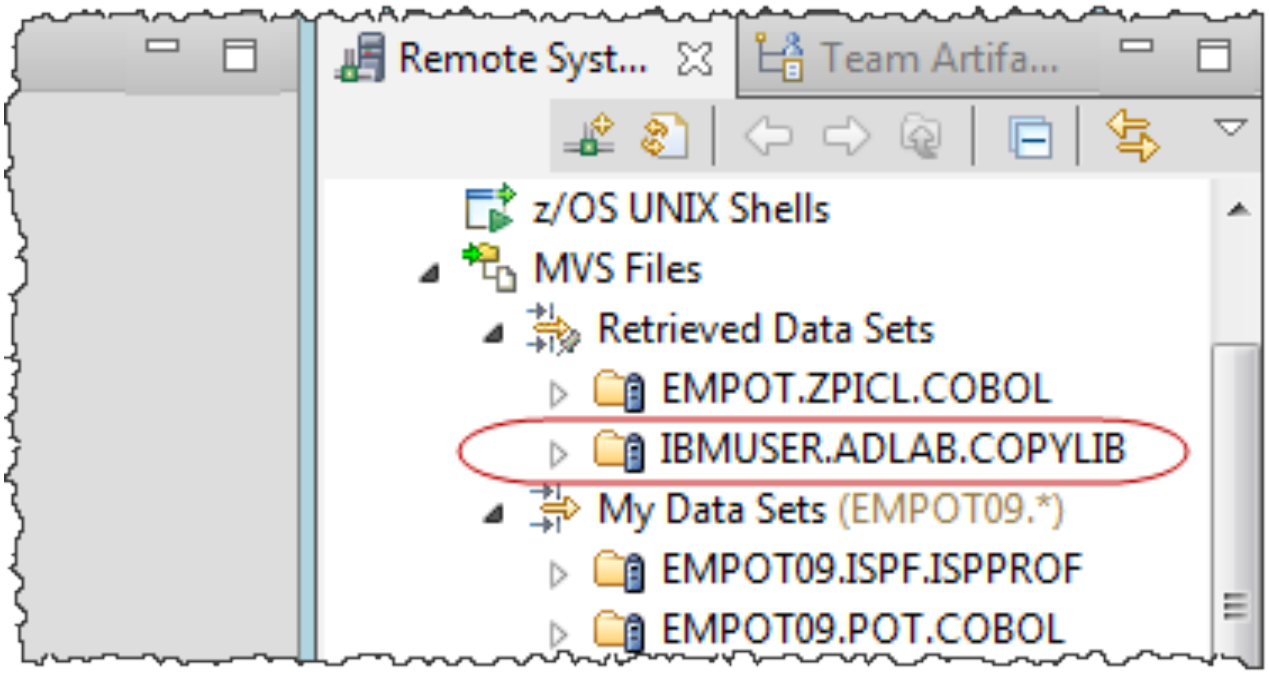
8.3.2 Type **IBMUSER.ADLAB.*** and press **Enter.**



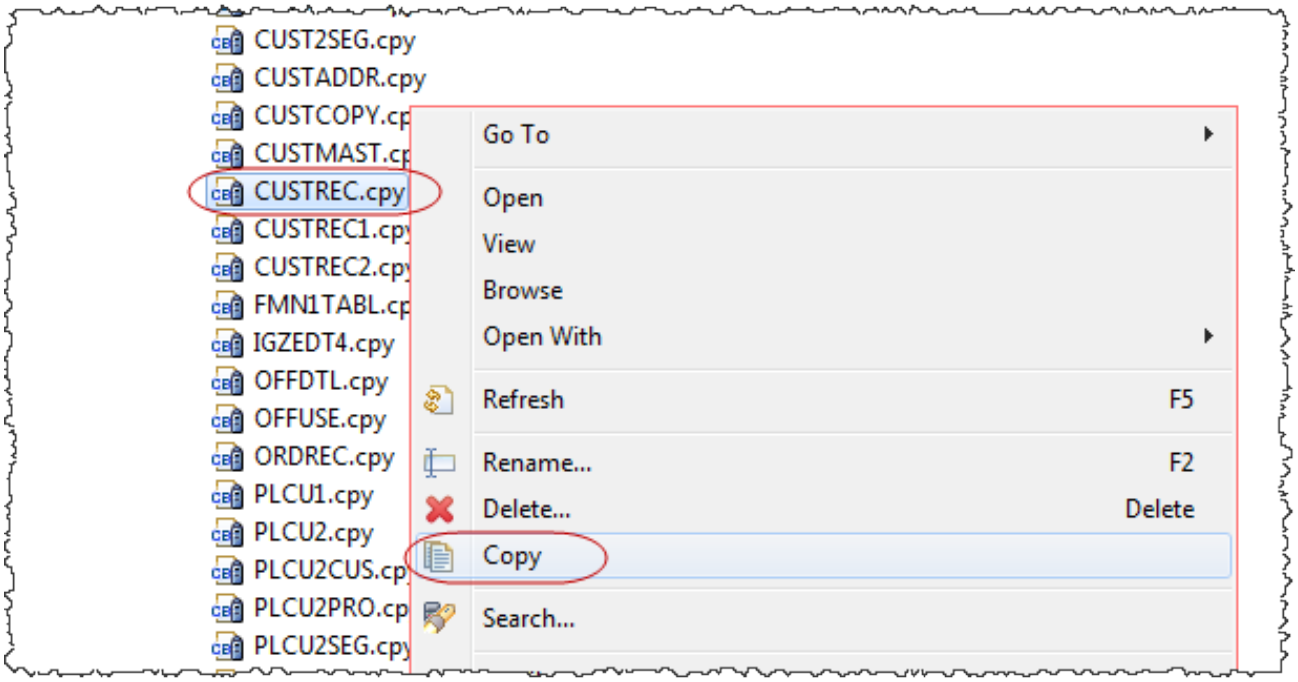
8.3.3 Select **IBMUSER.ADLAB.COPYLIB** and click **OK.**



8.3.4 The data set will be under *Retrieved Data Sets*.

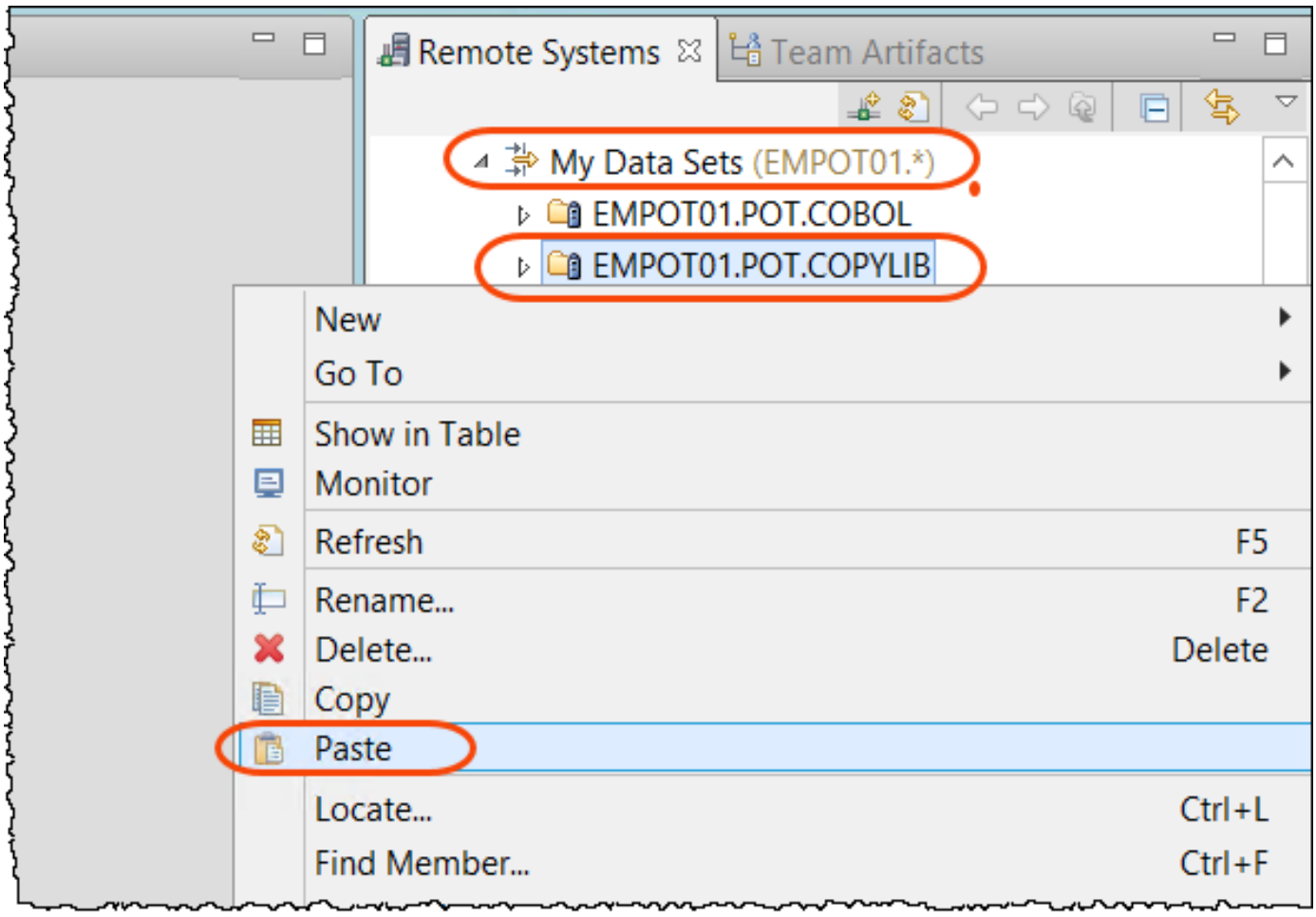


8.3.5 Expand **IBMUSER.ADLAB.COPYLIB**, right click on **CUSTREC.cpy** and select **Copy**.
You want to copy this member to your PDS.

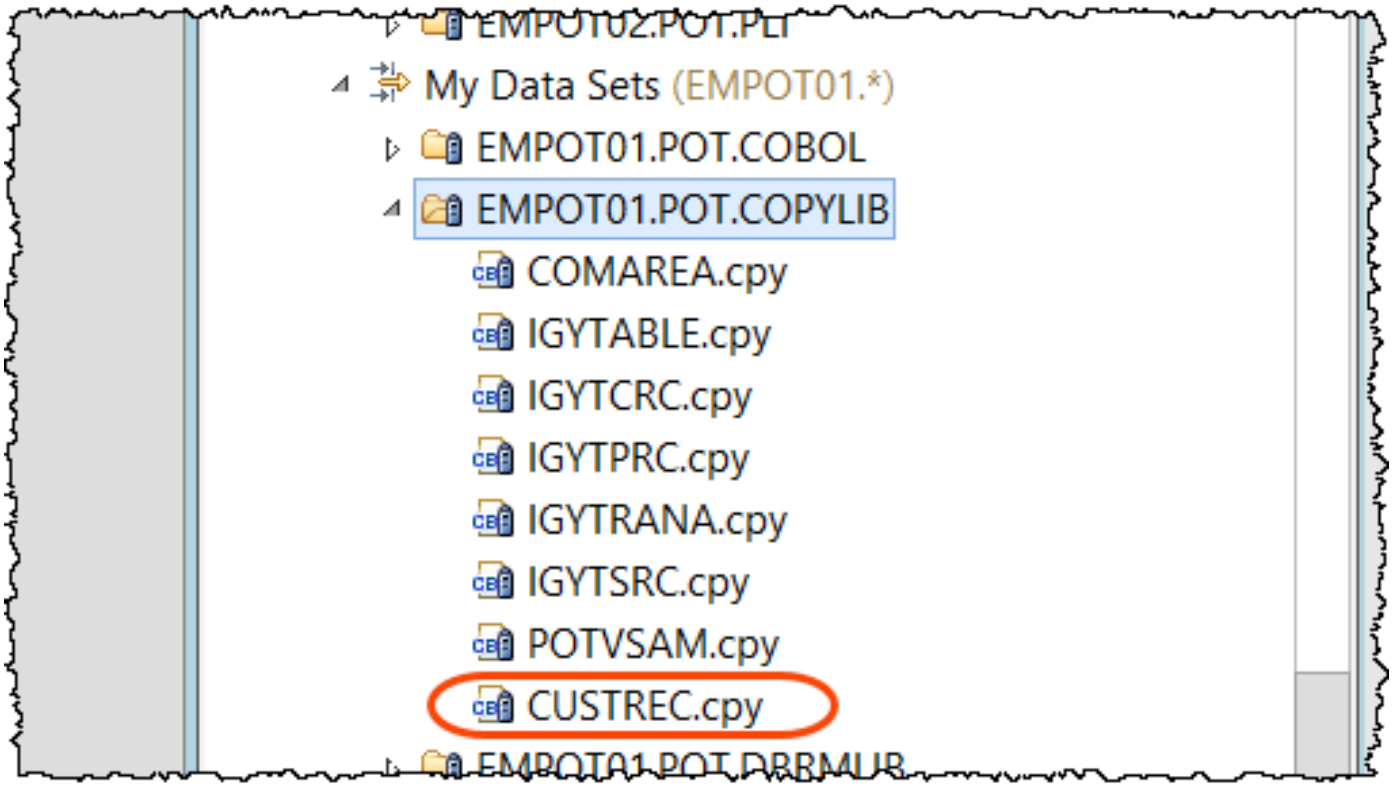


8.3.6 Navigate to **My Data Sets**, right click on **EMPOT01.POT.COPYLIB** and select **Paste**

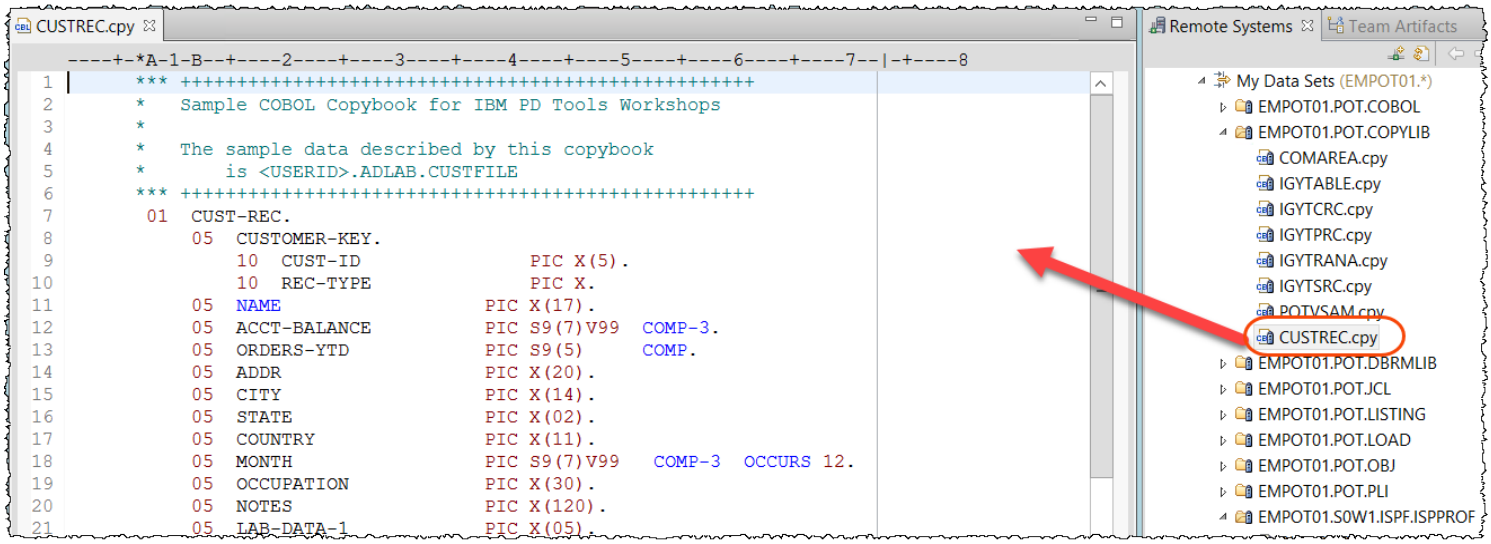
If this member is already there select **Overwrite**



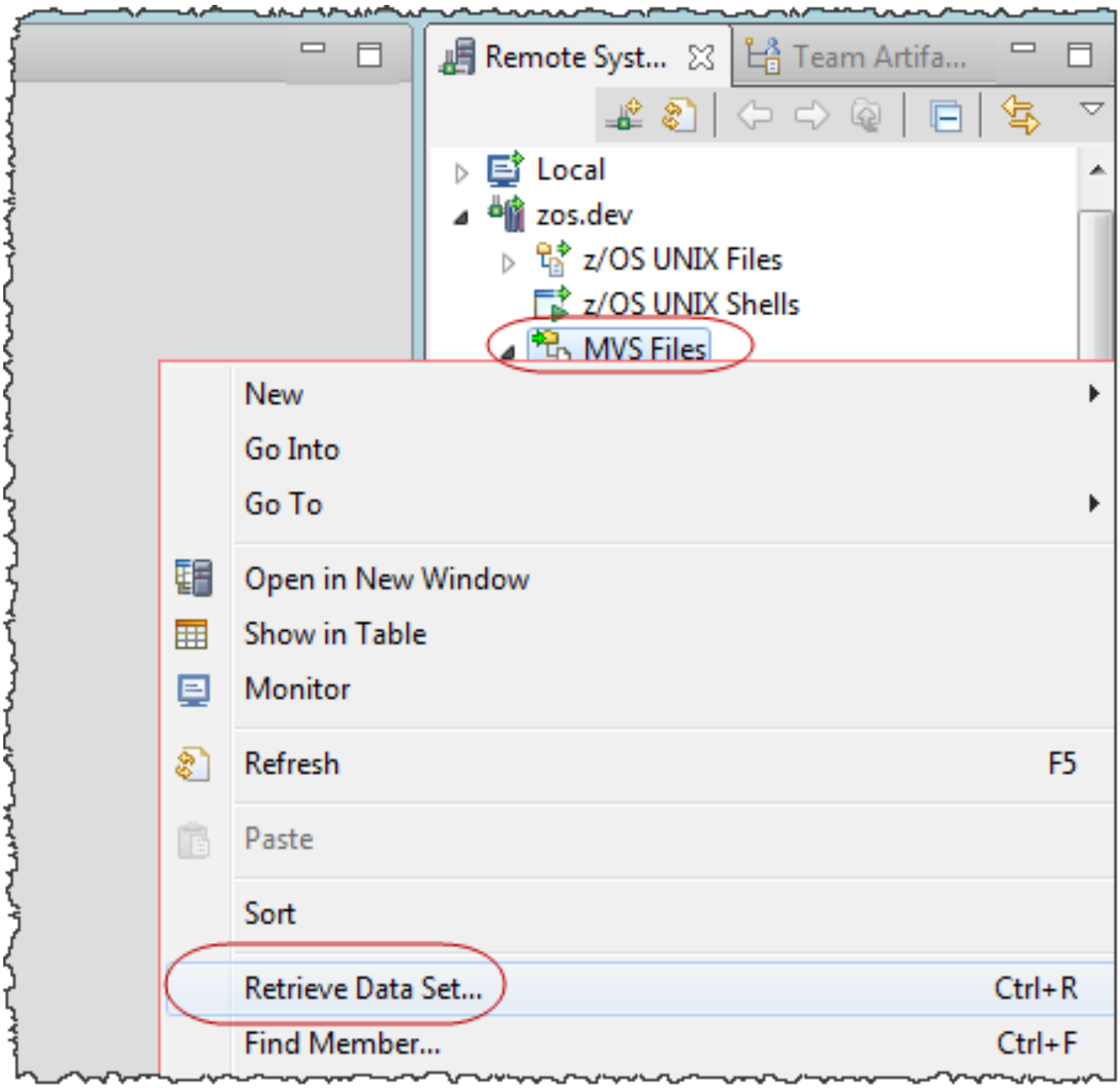
8.3.7 You should have:



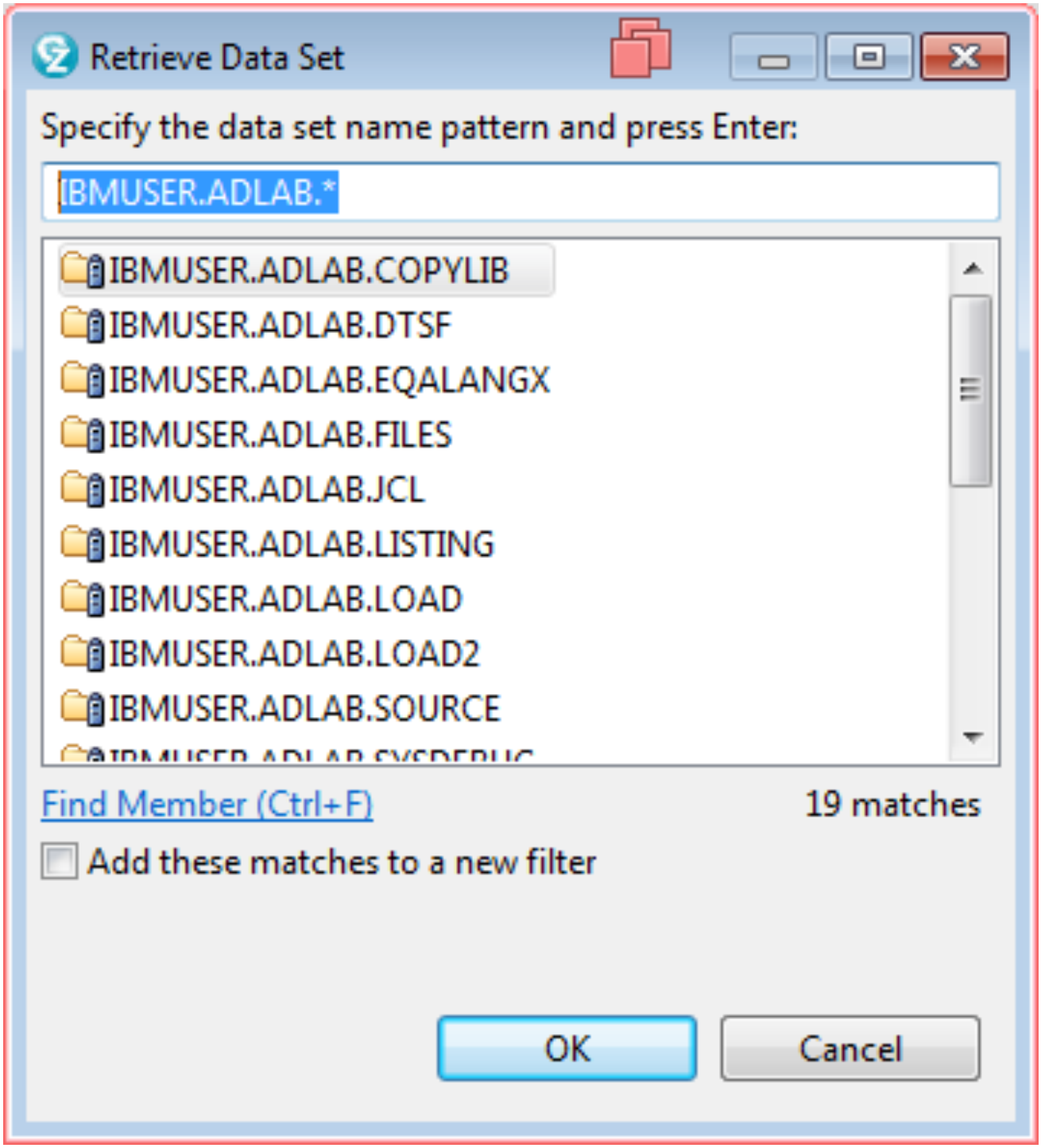
8.3.8 Double click on **CUSTREC.cpy** to edit it.
This copybook is the file layout of the file that you will see later. Note than some fields are COMP-3.



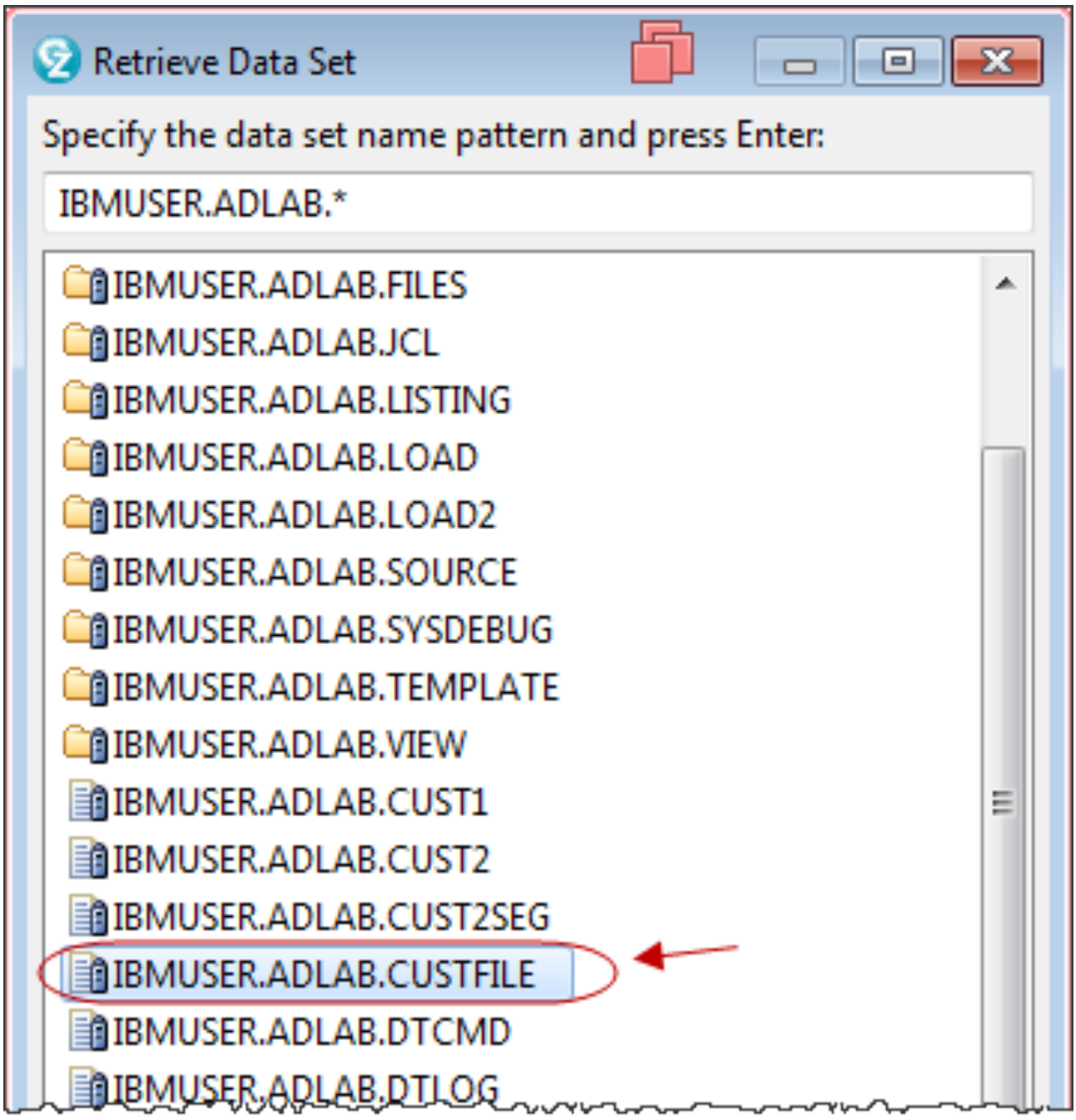
8.3.9 To see the data set, using the *Remote Systems* view, right click on **MVS Files** and select **Retrieve Data Set** ….
This is equivalent at the TSO 3.4 function.



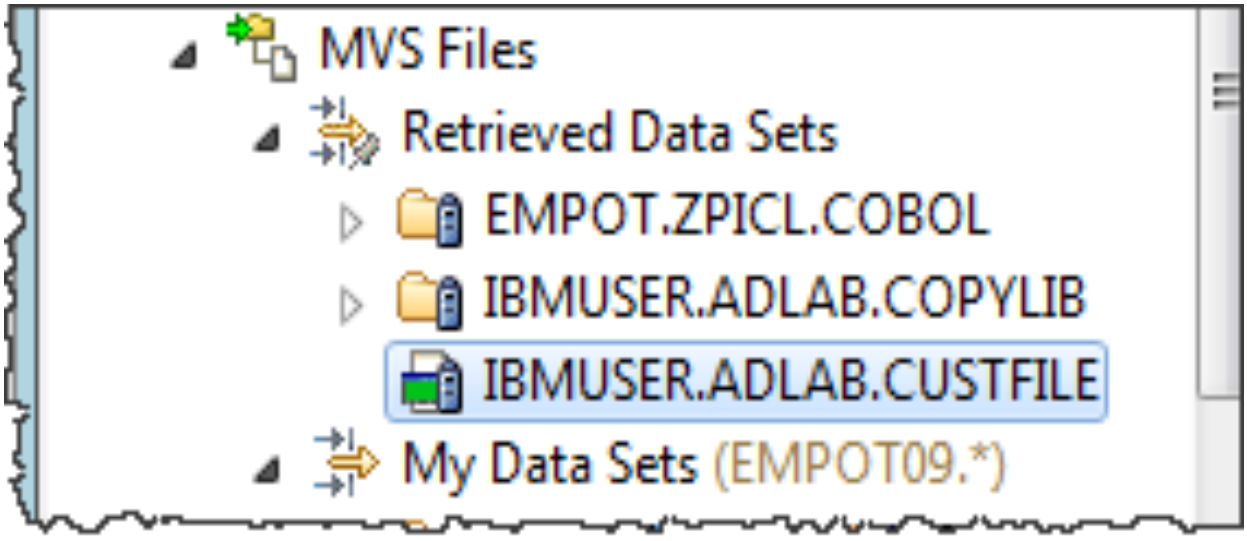
8.3.10 Type **IBMUSER.ADLAB.*** and press **Enter.**



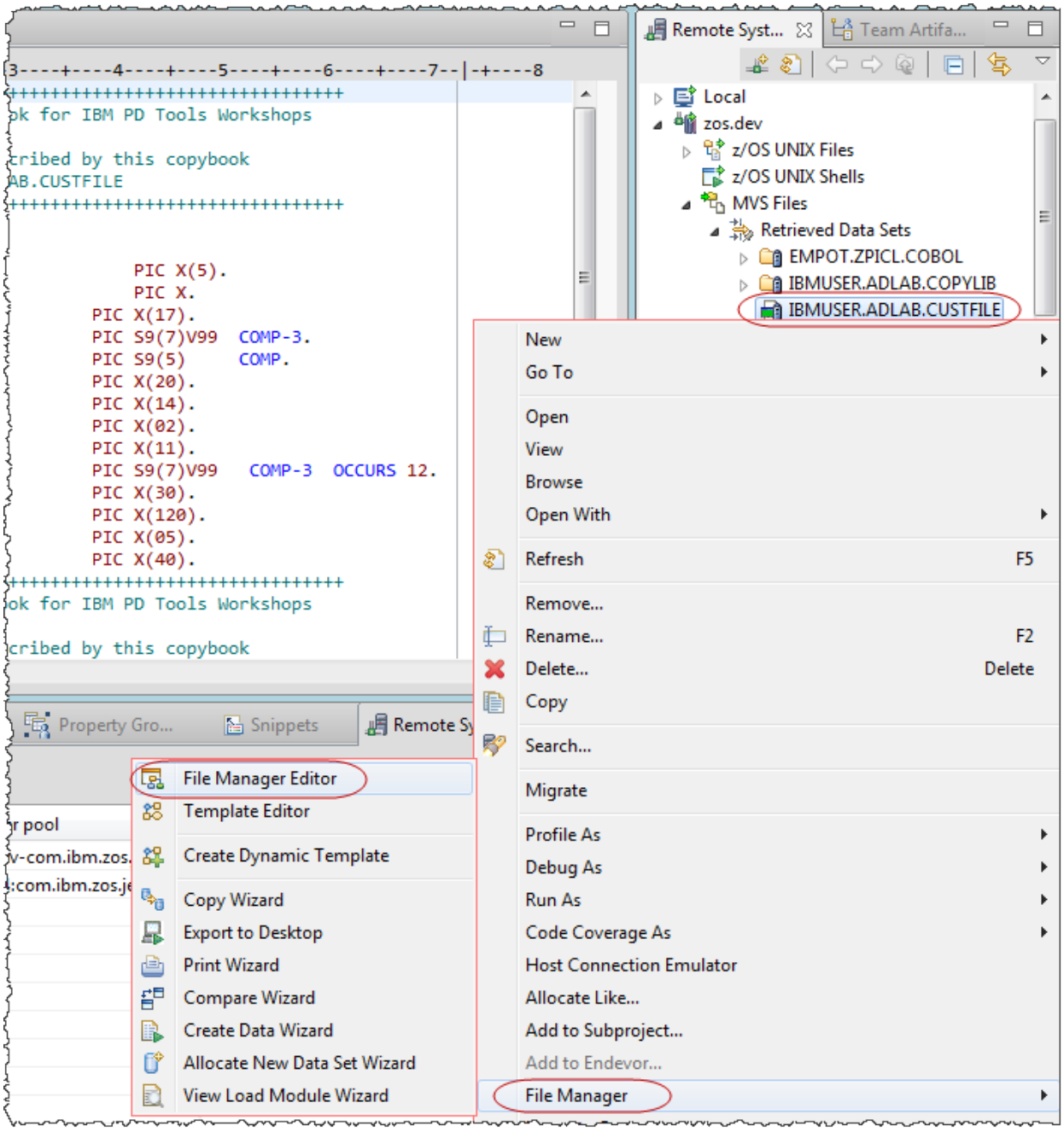
8.3.11 Scroll down, select **IBMUSER.ADLAB.CUSTFILE** and click **OK.**



8.3.12 The data set will be under *Retrieved Data Sets*.

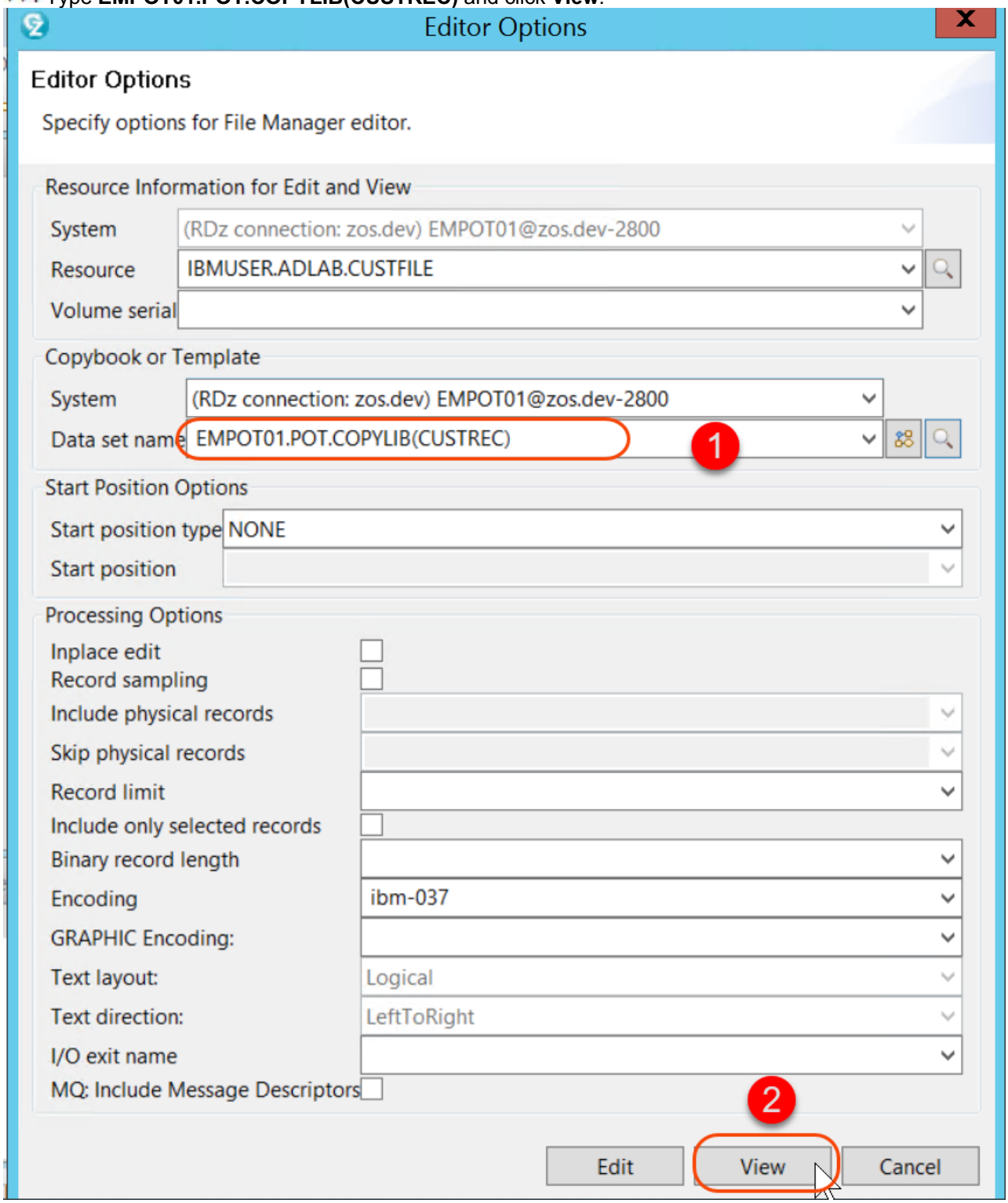


8.3.13 Right click on **IBMUSER.ADLAB.CUSTFILE** and select **File Manager -> File Manager Editor**. You want to map this file against the COBOL COPYBOOK that you copied.

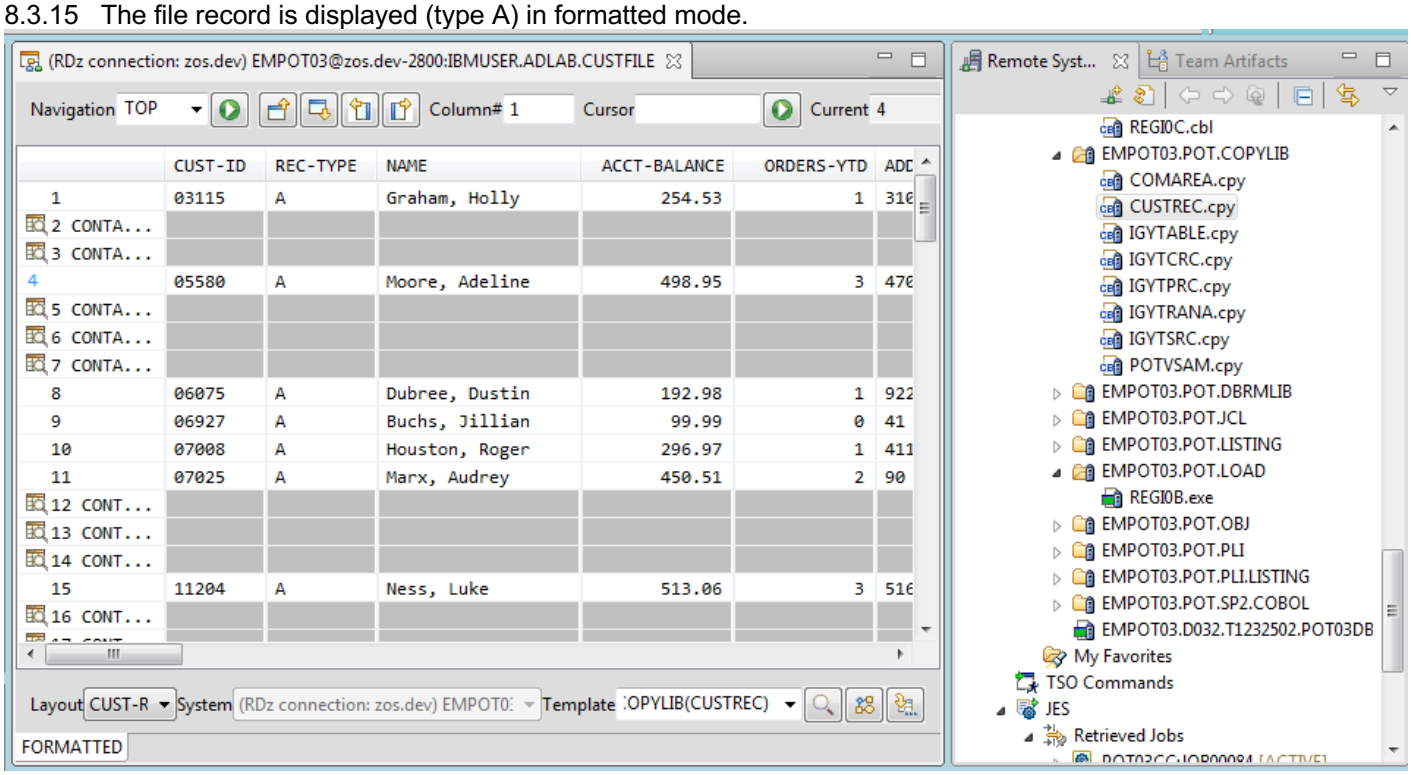

Remote Syst...
Team Artifa...
Local
zos.dev
z/OS UNIX Files
z/OS UNIX Shells
MVS Files
Retrieved Data Sets
EMPOT.ZPICL.COBOL
IBMUSER.ADLAB.COPYLIB
IBMUSER.ADLAB.CUSTFILE
PIC X(5).
PIC X.
PIC X(17).
PIC S9(7)V99 COMP-3.
PIC S9(5) COMP.
PIC X(20).
PIC X(14).
PIC X(02).
PIC X(11).
PIC S9(7)V99 COMP-3 OCCURS 12.
PIC X(30).
PIC X(120).
PIC X(05).
PIC X(40).
Property Gro...
Snippets
Remote Sy
File Manager Editor
Template Editor
Create Dynamic Template
Copy Wizard
Export to Desktop
Print Wizard
Compare Wizard
Create Data Wizard
Allocate New Data Set Wizard
View Load Module Wizard
New
Go To
Open
View
Browse
Open With
Refresh F5
Remove...
Rename... F2
Delete... Delete
Copy
Search...
Migrate
Profile As
Debug As
Run As
Code Coverage As
Host Connection Emulator
Allocate Like...
Add to Subproject...
Add to Endevor...
File Manager

8.3.14 On *Data set name* you will use the PDS and member that you have copied the COPYBOOK.

⏭ Type **EMPOT01.POT.COPYLIB(CUSTREC)** and click **View**.



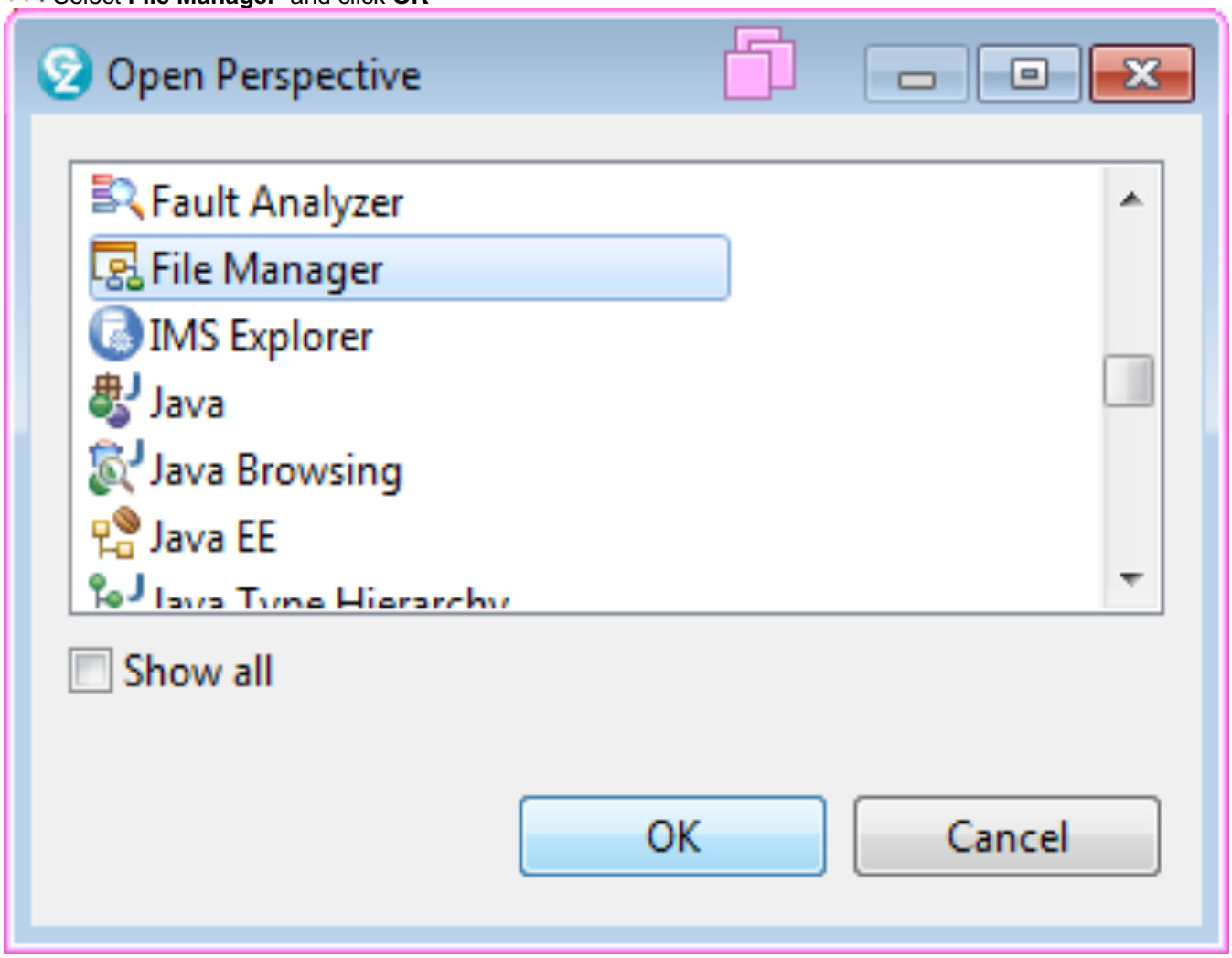
8.3.15 The file record is displayed (type A) in formatted mode.



8.3.16 We can see more details using the File manager perspective.

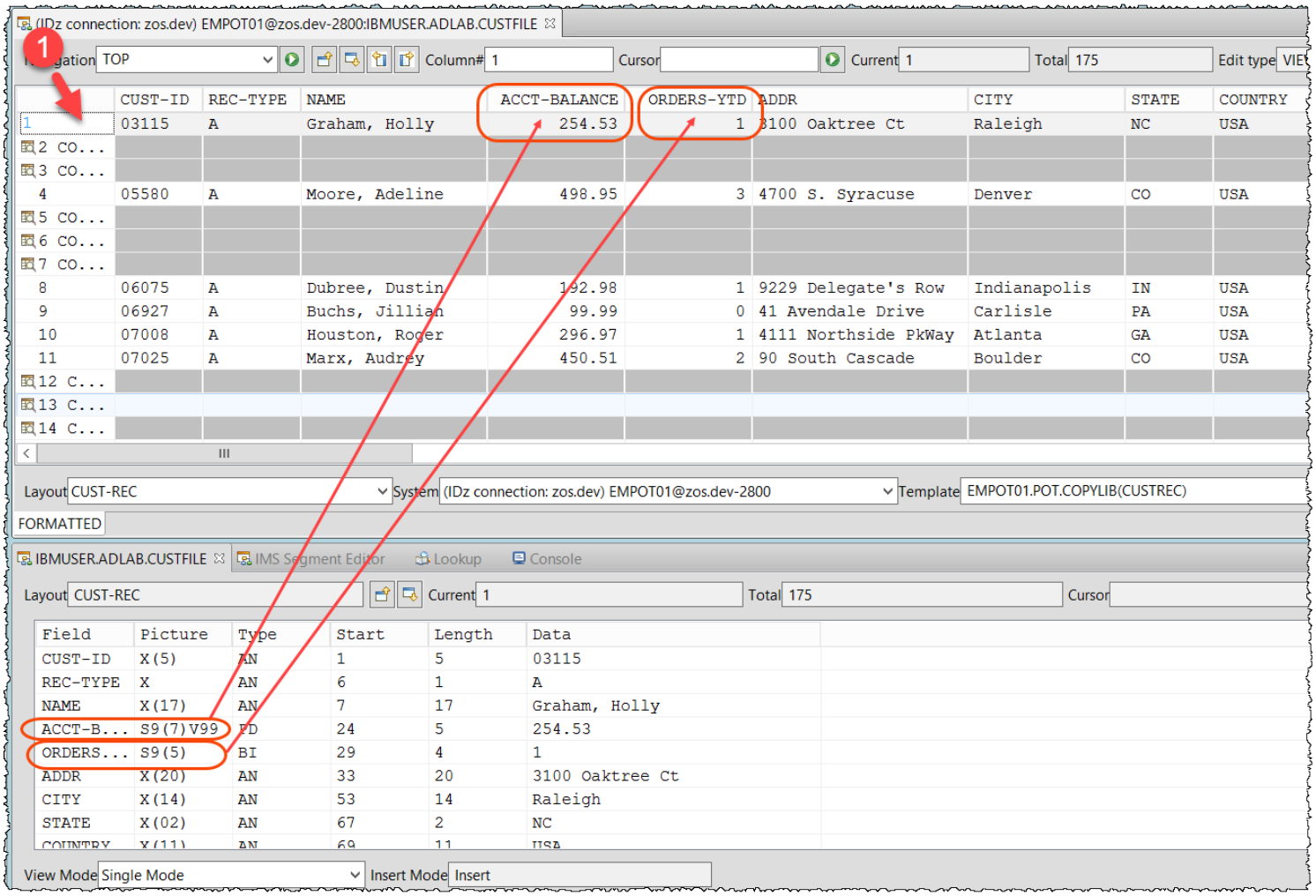
⏭ Select **Window > Perspective > Open Perspective > Other…**

⏭ Select **File Manager** and click **OK**



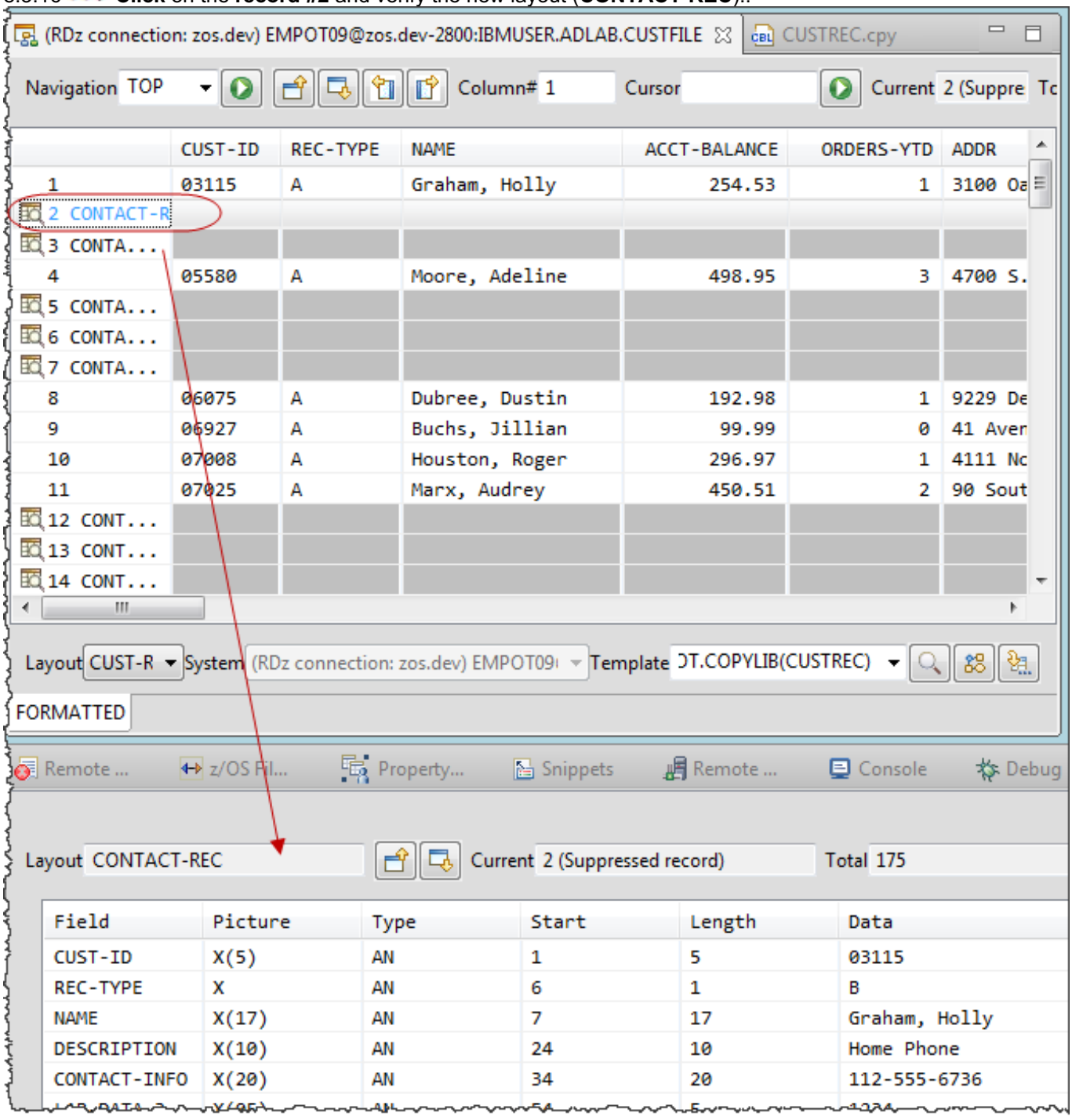
⏭ **Click** on the **record #1**.

Notice that the field **ACCT-BALANCE** that is **COMP-3** and **ORDERS-YTD** that is **COMP** are correctly displayed.

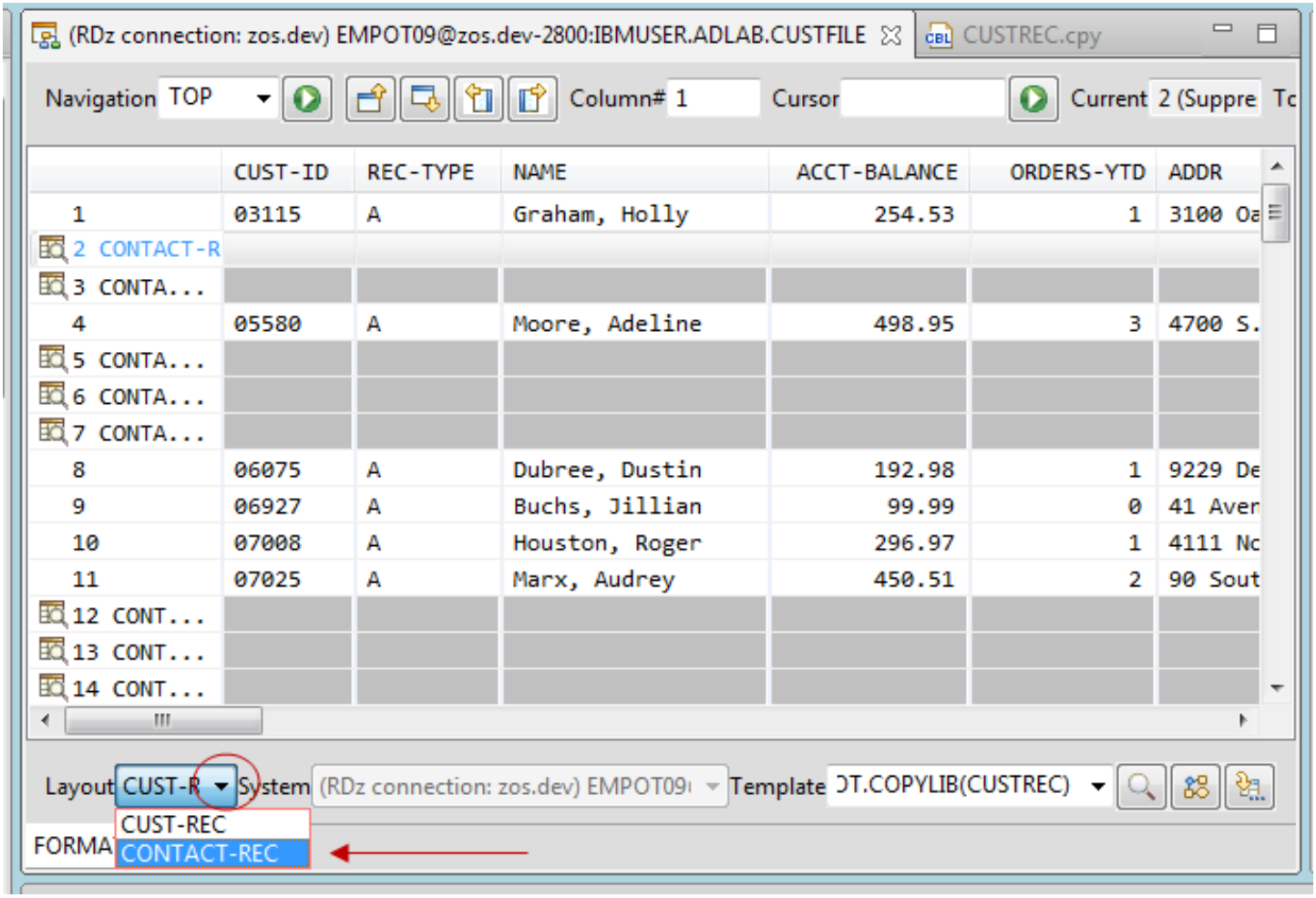

(IDz connection: zos.dev) EMPOT01@zos.dev-2800:IBMUSER.ADLAB.CUSTFILE
1
TOP
Column# 1
Cursor
Current 1
Total 175
Edit type
CUST-ID REC-TYPE NAME ACCT-BALANCE ORDERS-YTD ADDR CITY STATE COUNTRY
1 03115 A Graham, Holly 254.53 1 3100 Oaktree Ct Raleigh NC USA
2 CO...
3 CO...
4 05580 A Moore, Adeline 498.95 3 4700 S. Syracuse Denver CO USA
5 CO...
6 CO...
7 CO...
8 06075 A Dubree, Dustin 192.98 1 9229 Delegate's Row Indianapolis IN USA
9 06927 A Buchs, Jillian 99.99 0 41 Avendale Drive Carlisle PA USA
10 07008 A Houston, Roger 296.97 1 4111 Northside PkWay Atlanta GA USA
11 07025 A Marx, Audrey 450.51 2 90 South Cascade Boulder CO USA
12 C...
13 C...
14 C...
Layout CUST-REC
System (IDz connection: zos.dev) EMPOT01@zos.dev-2800
Template EMPOT01.POT.COPYLIB(CUSTREC)
FORMATTED
IBMUSER.ADLAB.CUSTFILE
IMS Segment Editor
Lookup
Console
Layout CUST-REC
Current 1
Total 175
Cursor
Field Picture Type Start Length Data
CUST-ID X(5) AN 1 5 03115
REC-TYPE X AN 6 1 A
NAME X(17) AN 7 17 Graham, Holly
ACCT-B... S9(7)V99 PD 24 5 254.53
ORDERS... S9(5) BI 29 4 1
ADDR X(20) AN 33 20 3100 Oaktree Ct
CITY X(14) AN 53 14 Raleigh
STATE X(02) AN 67 2 NC
COUNTRY X(11) AN 69 11 USA
View Mode Single Mode
Insert Mode Insert

You may have to scroll down this view to see the formatted field names.
Also notice that there are two views of the data. The multiple record view appears at the top by default, and the single record view appears at the bottom and displays only one record at a time.
If you select (click) a record in the multiple-record view that record displays in the single record view.

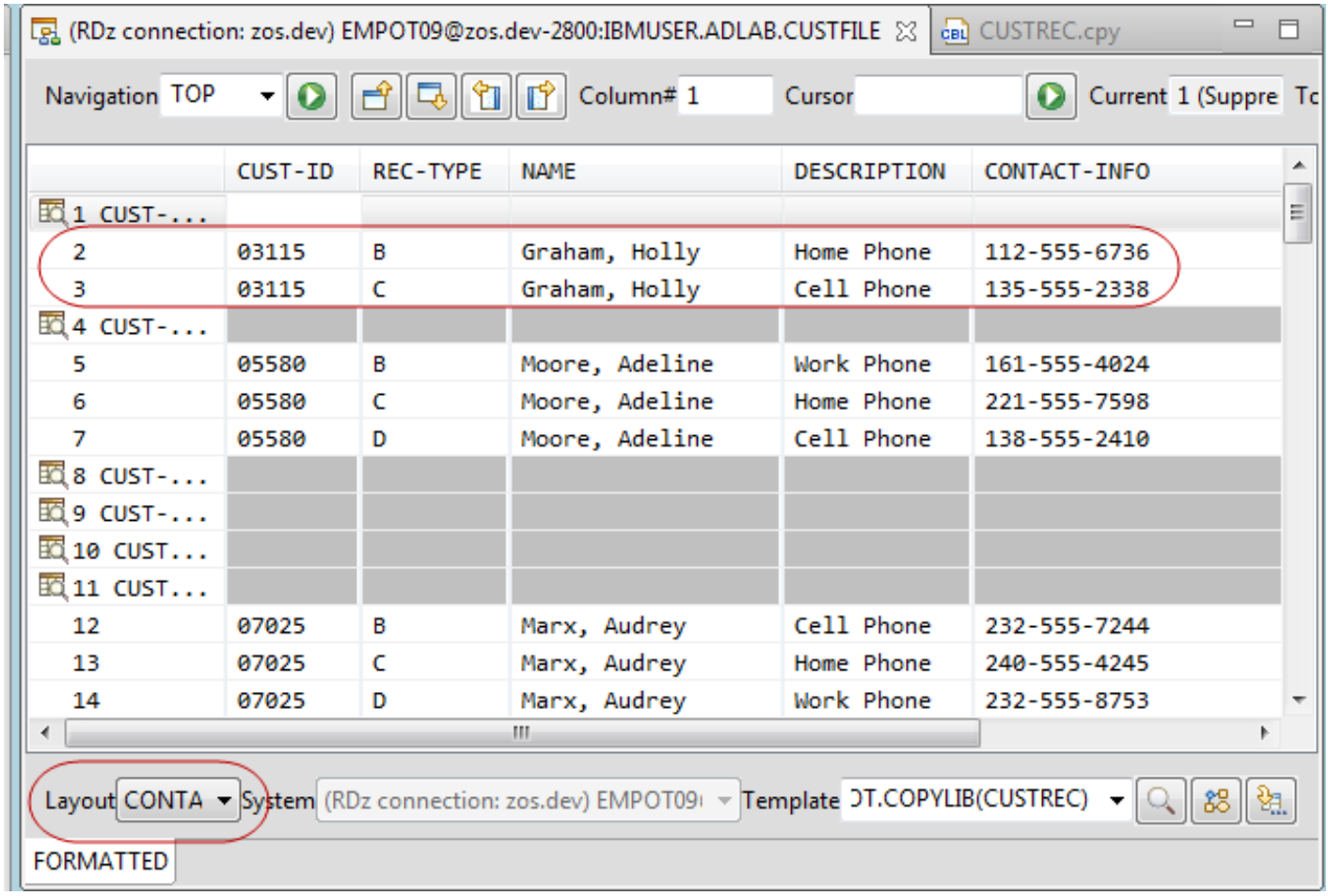
8.3.16 **Click** on the **record #2** and verify the new layout (**CONTACT-REC**)..



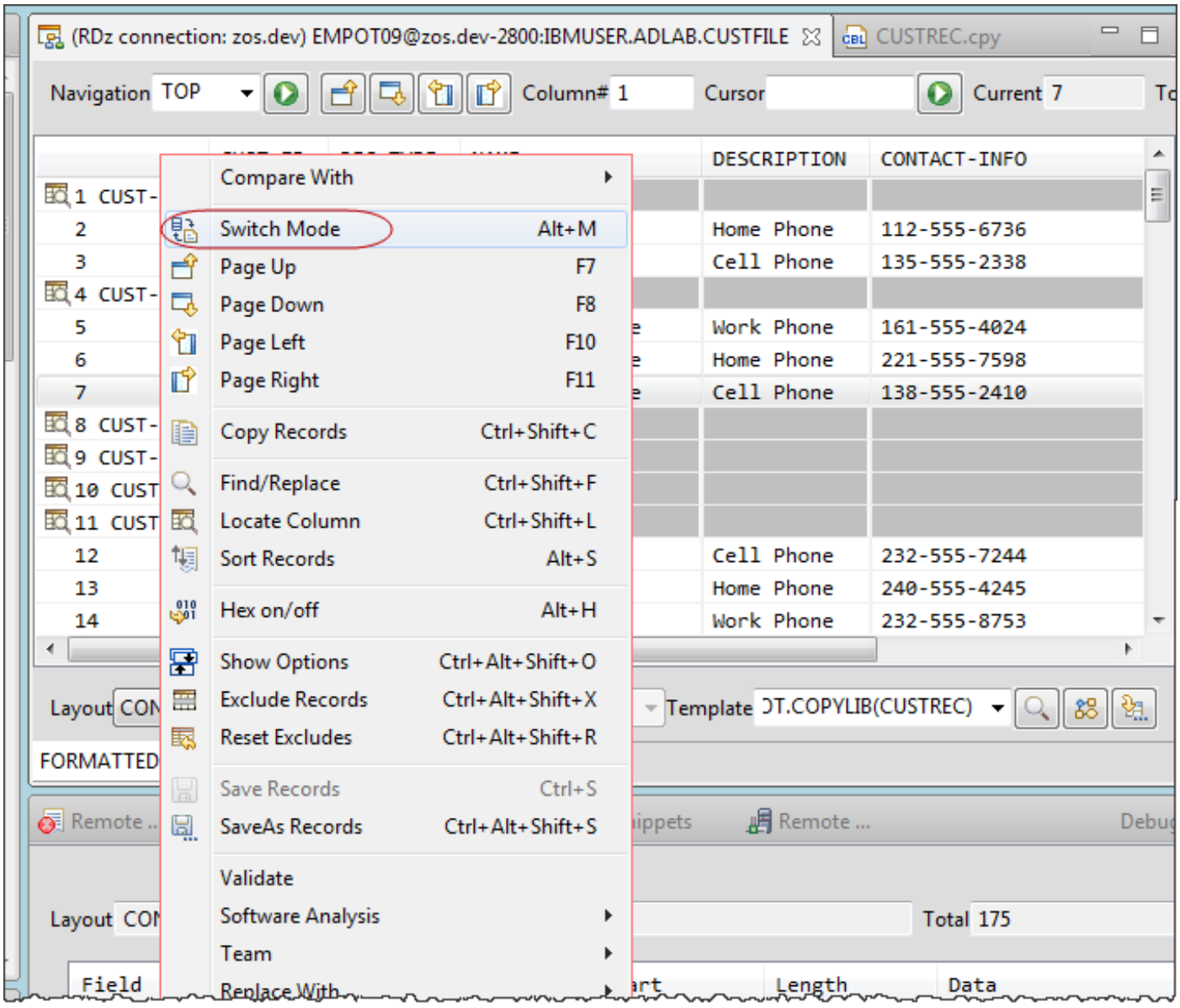
8.3.17 Click on the *Layout* pull-down in the editor and select **CONTACT-REC**. As you see, you can work with multiple layouts.


(RDz connection: zos.dev) EMPOT09@zos.dev-2800:IBMUSER.ADLAB.CUSTFILE
CUSTREC.cpy
Navigation TOP
Column# 1
Cursor
Current 2 (Suppre
CUST-ID REC-TYPE NAME ACCT-BALANCE ORDERS-YTD ADDR
1 03115 A Graham, Holly 254.53 1 3100 Oa
2 CONTACT-R
3 CONTA...
4 05580 A Moore, Adeline 498.95 3 4700 S.
5 CONTA...
6 CONTA...
7 CONTA...
8 06075 A Dubree, Dustin 192.98 1 9229 De
9 06927 A Buchs, Jillian 99.99 0 41 Aver
10 07008 A Houston, Roger 296.97 1 4111 Nc
11 07025 A Marx, Audrey 450.51 2 90 Sout
12 CONT...
13 CONT...
14 CONT...
Layout CUST-R
System (RDz connection: zos.dev) EMPOT09
Template)T.COPYLIB(CUSTREC)
CUST-REC
CONTACT-REC
FORMA

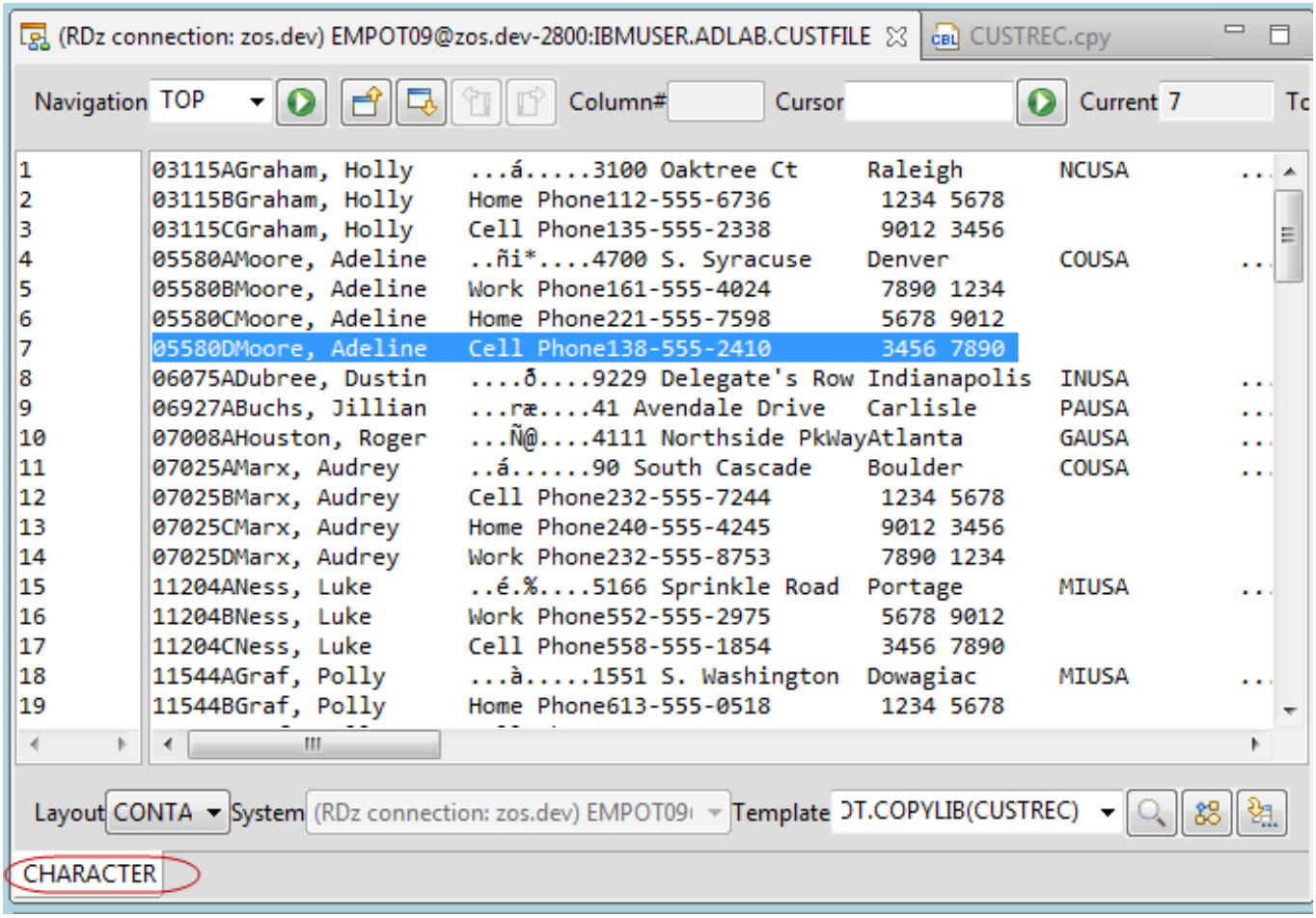
8.3.18 You now can see that records 2 and 3 are displayed (type B and C) in formatted mode.



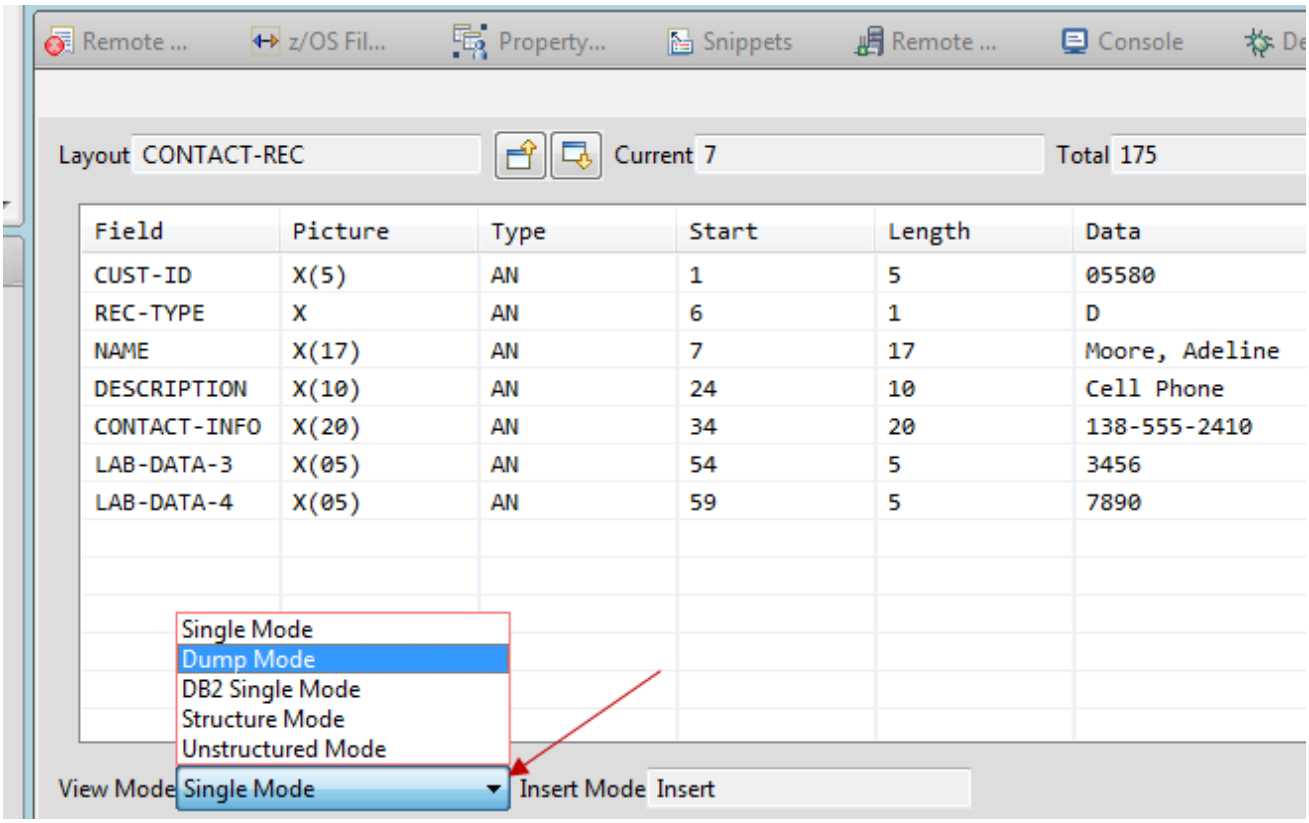
8.3.19 Right click on the editor and verify that there are many options.
Click on **Switch Mode**

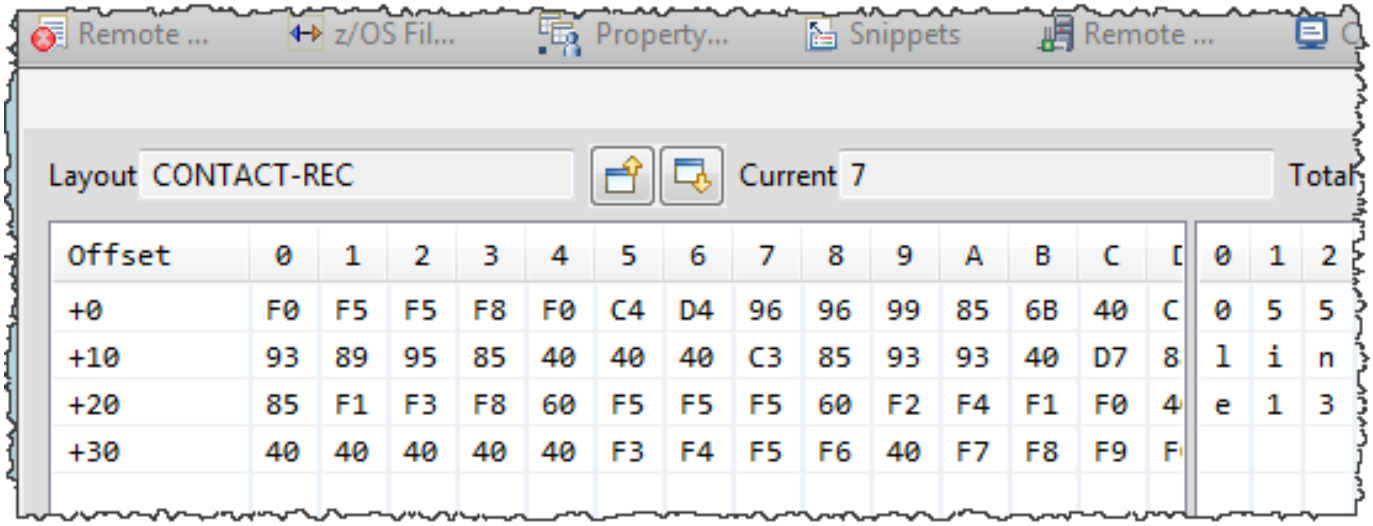

(RDz connection: zos.dev) EMPOT09@zos.dev-2800:IBMUSER.ADLAB.CUSTFILE
CUSTREC.cpy
Navigation TOP
Column# 1
Cursor
Current 7
DESCRIPTION
CONTACT-INFO
Home Phone 112-555-6736
Cell Phone 135-555-2338
Work Phone 161-555-4024
Home Phone 221-555-7598
Cell Phone 138-555-2410
Cell Phone 232-555-7244
Home Phone 240-555-4245
Work Phone 232-555-8753
Compare With
Switch Mode Alt+M
Page Up F7
Page Down F8
Page Left F10
Page Right F11
Copy Records Ctrl+Shift+C
Find/Replace Ctrl+Shift+F
Locate Column Ctrl+Shift+L
Sort Records Alt+S
Hex on/off Alt+H
Show Options Ctrl+Alt+Shift+O
Exclude Records Ctrl+Alt+Shift+X
Reset Excludes Ctrl+Alt+Shift+R
Save Records Ctrl+S
SaveAs Records Ctrl+Alt+Shift+S
Validate
Software Analysis
Team
Replace With
Layout
Template
FORMATTED
Total 175
Field
Length
Data

8.3.20 The multiple-record view display is changed to character mode. In character mode, the data is not formatted according to the fields in layout.



8.3.21 On the lower view notice the **View Mode** options.
Use the drop down and select **Dump Mode:**





There are lots of capabilities here.. You can try play a bit, but due to time constrains we will not cover all.

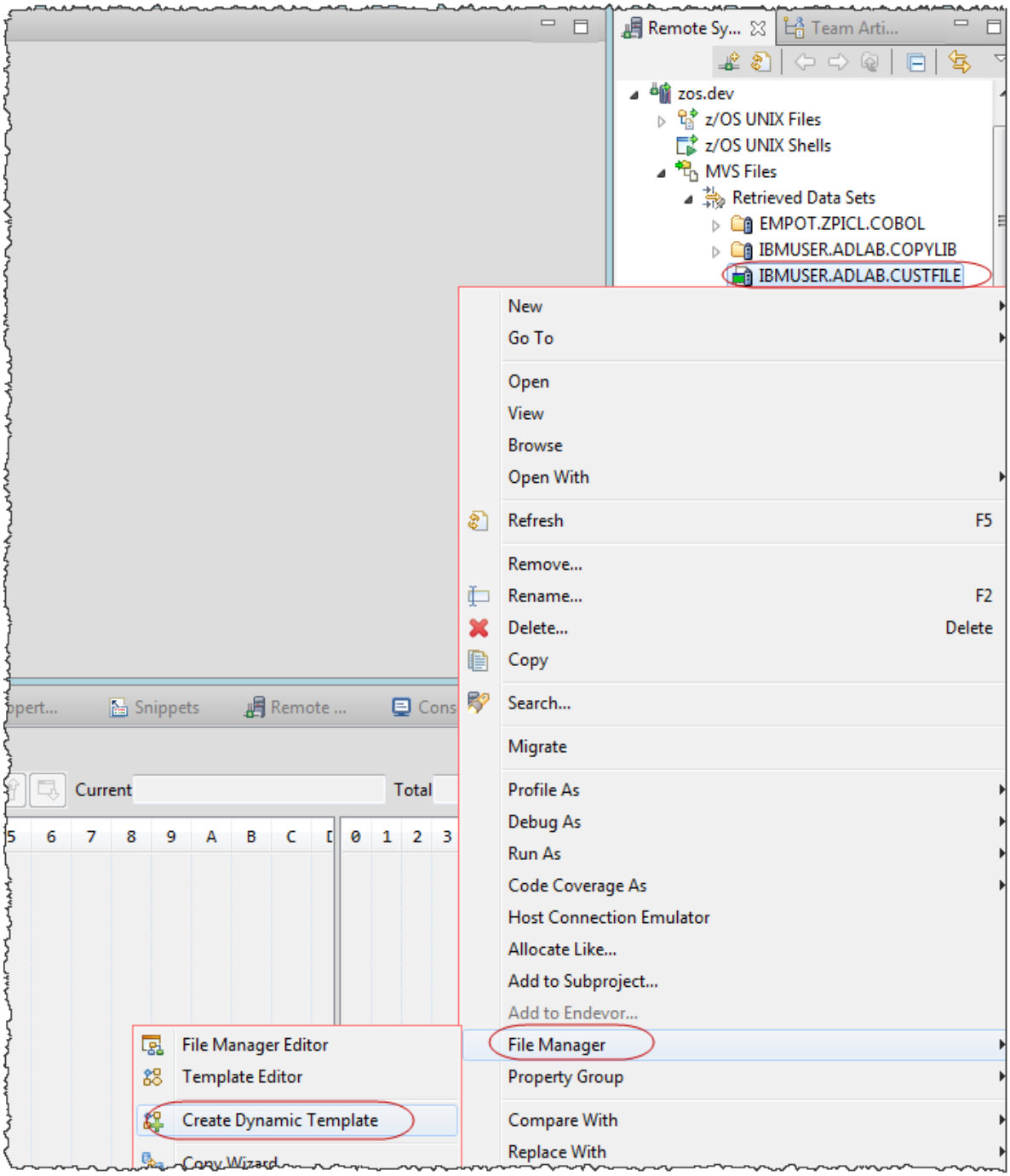
8.3.22 ⏭ Use **Ctrl + Shift + F4** to close all opened editors.

___ 8.4 Working with templates

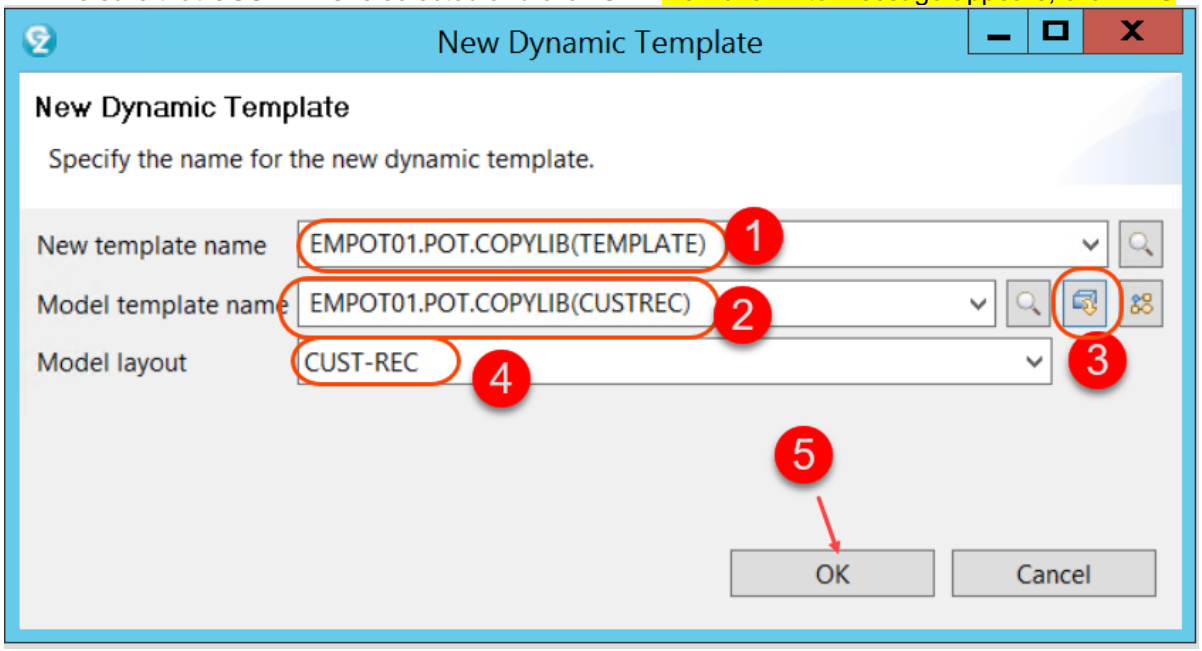
When you use the File Manager editor or viewer, you can specify a layout in the form of either a copybook or a template. If you specify a copybook, the editor/viewer can display records formatted according to the fields in the copybook. While a copybook defines the record layout, it cannot be used to select only a subset of records. A template, like a copybook, also has fields and defines the record layout. In addition, in a template you can specify:
- Record selection criteria (so that only the selected records will display)
- Field selection (so that only selected fields will display)
- Formatting of individual fields (for example, to always display a certain field in hexadecimal)
- And other formatting and data manipulation settings

You can use an existing copybook as the basis for a new template. All of the fields in the copybook are copied into the new template, and then you save the template. Templates are stored in PDS or library data sets. After you have saved a template, you can re-use it again whenever you need it. Templates can be used by other File Manager utilities other than just the editor and viewer.
For example, if you have a template that selects records, you can use it with the File Manager copy utility, and only the selected records will be copied.

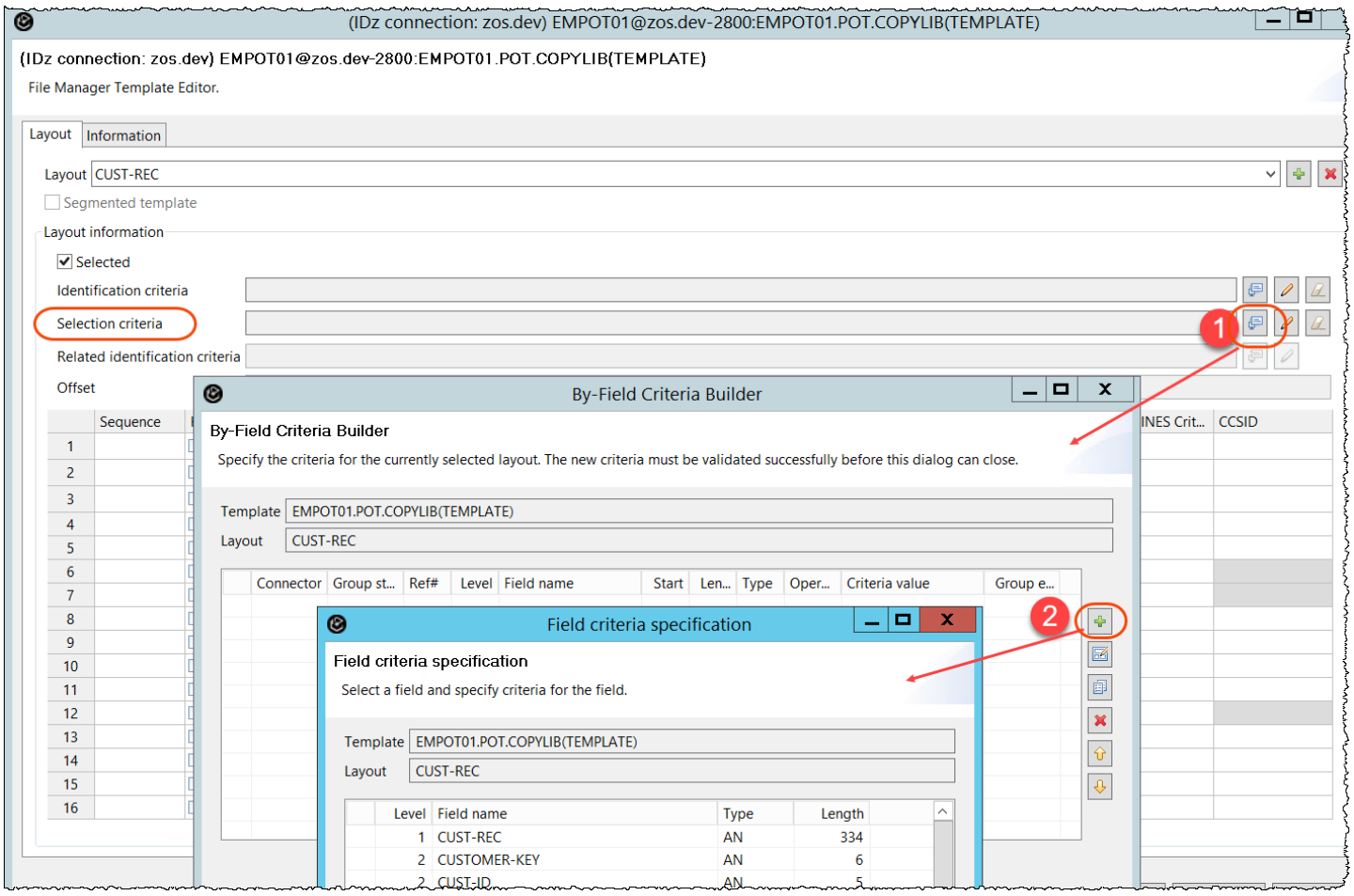
8.4.1  Go back to **z/OS Projects** Perspective and using *Remote Systems Explorer* view, right click **IBMUSER.ADLAB.CUSTFILE** file and select **File Manager -> Create Dynamic Template**


Remote Sy...
Team Arti...
zos.dev
z/OS UNIX Files
z/OS UNIX Shells
MVS Files
Retrieved Data Sets
EMPOT.ZPICL.COBOL
IBMUSER.ADLAB.COPYLIB
IBMUSER.ADLAB.CUSTFILE
New
Go To
Open
View
Browse
Open With
Refresh F5
Remove...
Rename... F2
Delete... Delete
Copy
Search...
Migrate
Profile As
Debug As
Run As
Code Coverage As
Host Connection Emulator
Allocate Like...
Add to Subproject...
Add to Endevor...
File Manager
Property Group
Compare With
Replace With
Snippets
Remote ...
Current
Total
File Manager Editor
Template Editor
Create Dynamic Template
Copy Wizard

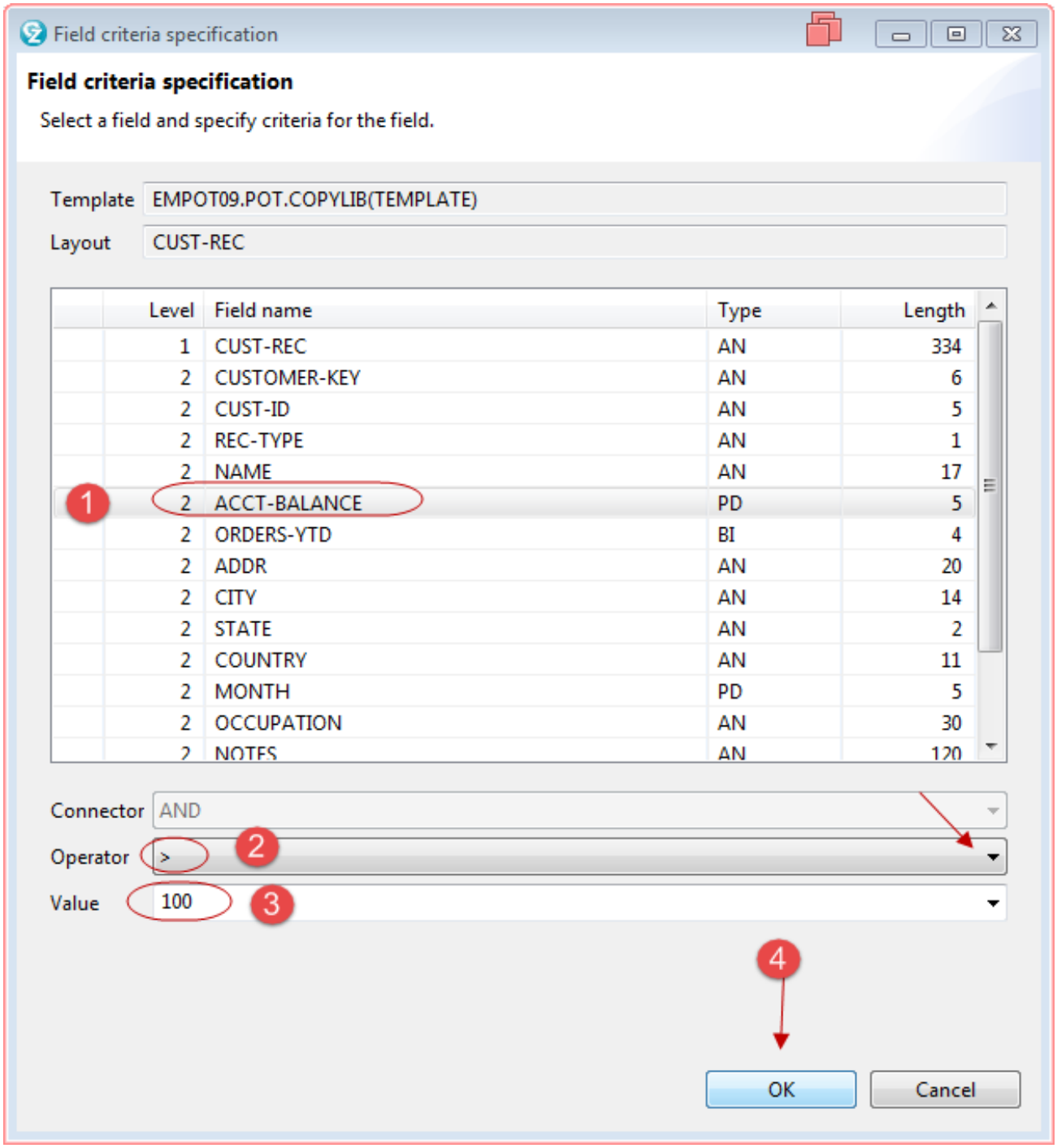
8.4.2 As *New template name* type **EMPOT01.POT.COPYLIB(TEMPLATE)**
For *Model template name* type **EMPOT01.POT.COPYLIB(CUSTREC)** and click on icon (**Load model template**).
Be sure that **CUST-REC** is selected and click **OK.** If an overwrite message appears, click **YES**.



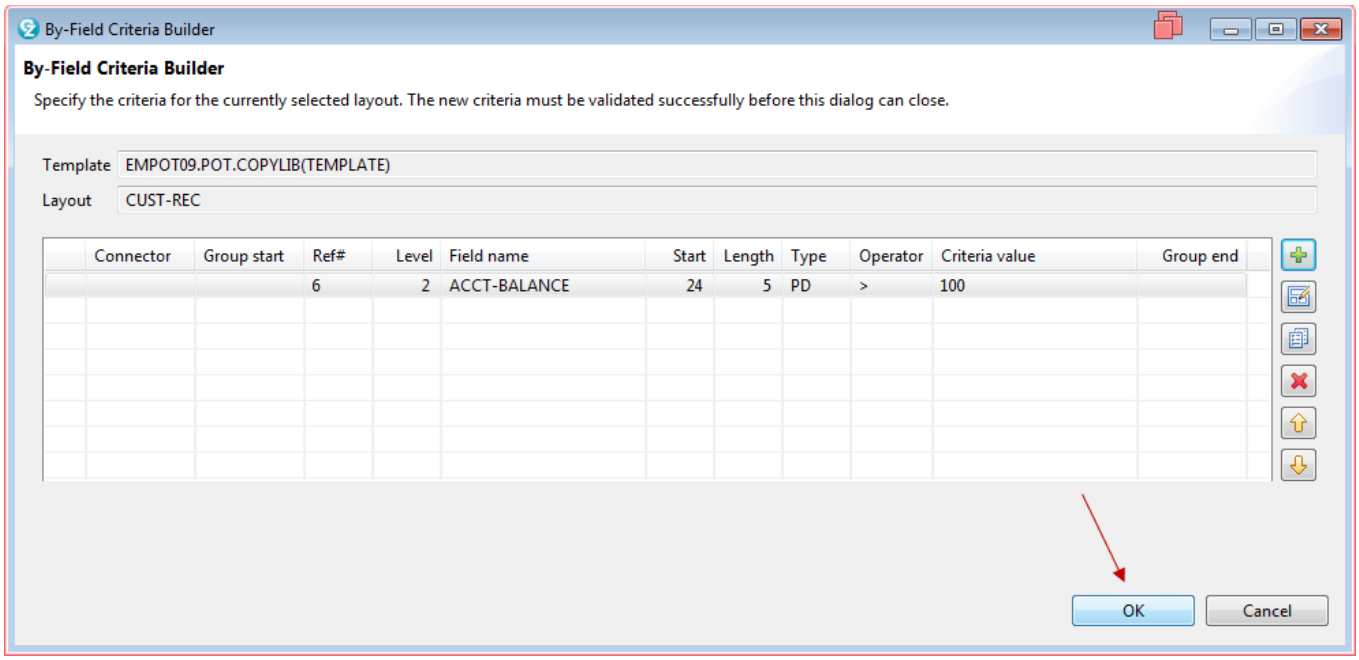
8.4.3 On the *File Manager Template Editor* you will add a selection criteria.
You will select all customers that have the ACCT-BALANCE **bigger than 100** and we want to display only **CUST-ID, NAME** and **ACCT-BALANCE**.
1 On ***Selection criteria*** click on the icon .
2 Using **By-Field Criteria Builder** click on the icon .


(IDz connection: zos.dev) EMPOT01@zos.dev-2800:EMPOT01.POT.COPYLIB(TEMPLATE)
(IDz connection: zos.dev) EMPOT01@zos.dev-2800:EMPOT01.POT.COPYLIB(TEMPLATE)
File Manager Template Editor.
Layout
Information
Layout CUST-REC
Segmented template
Layout information
Selected
Identification criteria
Selection criteria
Related identification criteria
Offset
Sequence
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
By-Field Criteria Builder
By-Field Criteria Builder
Specify the criteria for the currently selected layout. The new criteria must be validated successfully before this dialog can close.
Template EMPOT01.POT.COPYLIB(TEMPLATE)
Layout CUST-REC
Connector
Group st...
Ref#
Level
Field name
Start
Len...
Type
Oper...
Criteria value
Group e...
INES Crit...
CCSID
Field criteria specification
Field criteria specification
Select a field and specify criteria for the field.
Template EMPOT01.POT.COPYLIB(TEMPLATE)
Layout CUST-REC
Level
Field name
Type
Length
1 CUST-REC AN 334
2 CUSTOMER-KEY AN 6
2 CUST-ID AN 5
1
2

8.4.4 On the *Field criteria specification* select **ACCT-BALANCE**
On the *Operator* using the drop down select **>** and on the *Value* type **100** and click **OK**.

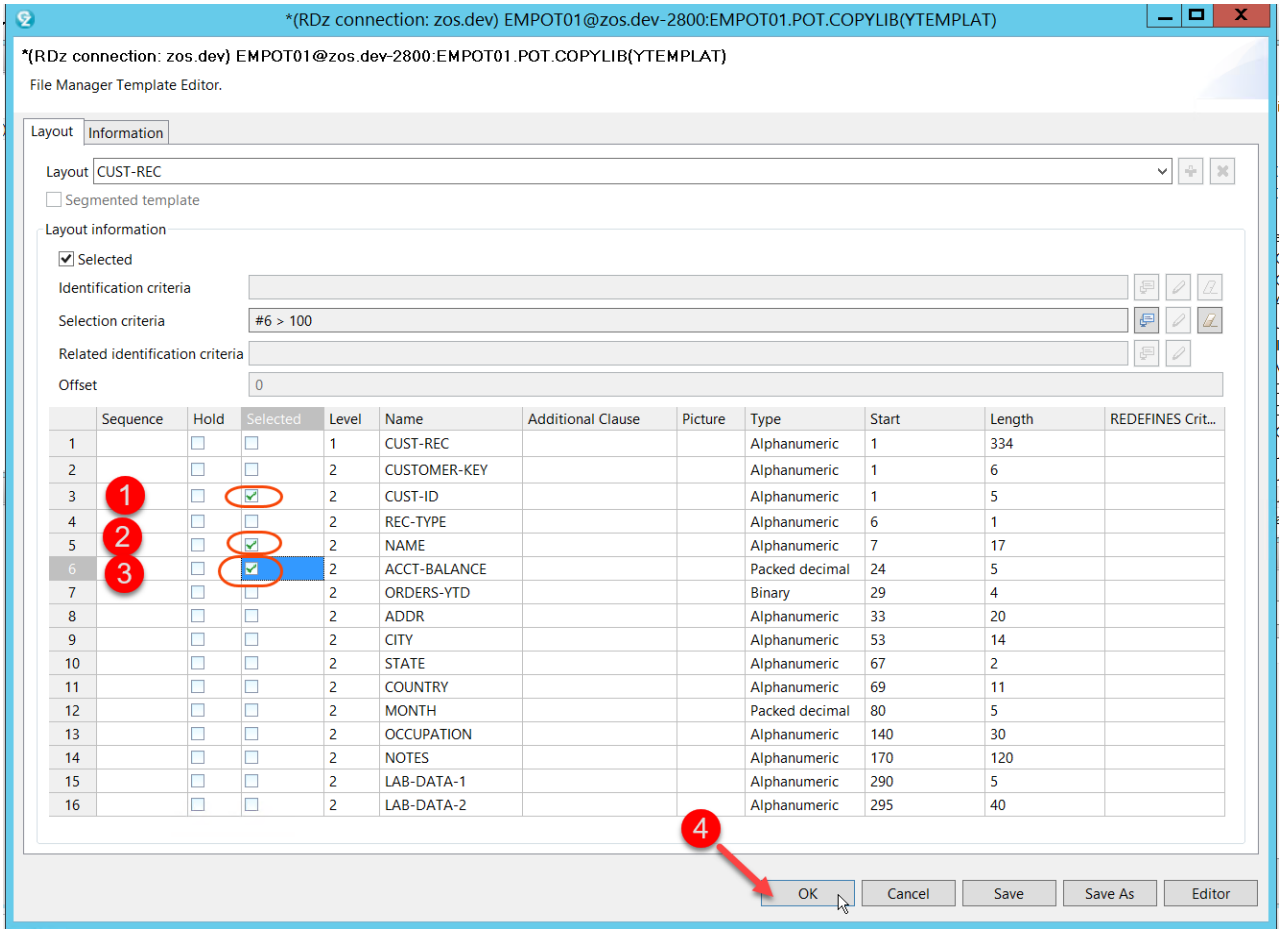


8.4.5 The result is shown below..
Click **OK**


By-Field Criteria Builder
By-Field Criteria Builder
Specify the criteria for the currently selected layout. The new criteria must be validated successfully before this dialog can close.
Template EMPOT09.POT.COPYLIB(TEMPLATE)
Layout CUST-REC
Connector Group start Ref# Level Field name Start Length Type Operator Criteria value Group end
6 2 ACCT-BALANCE 24 5 PD > 100
OK
Cancel

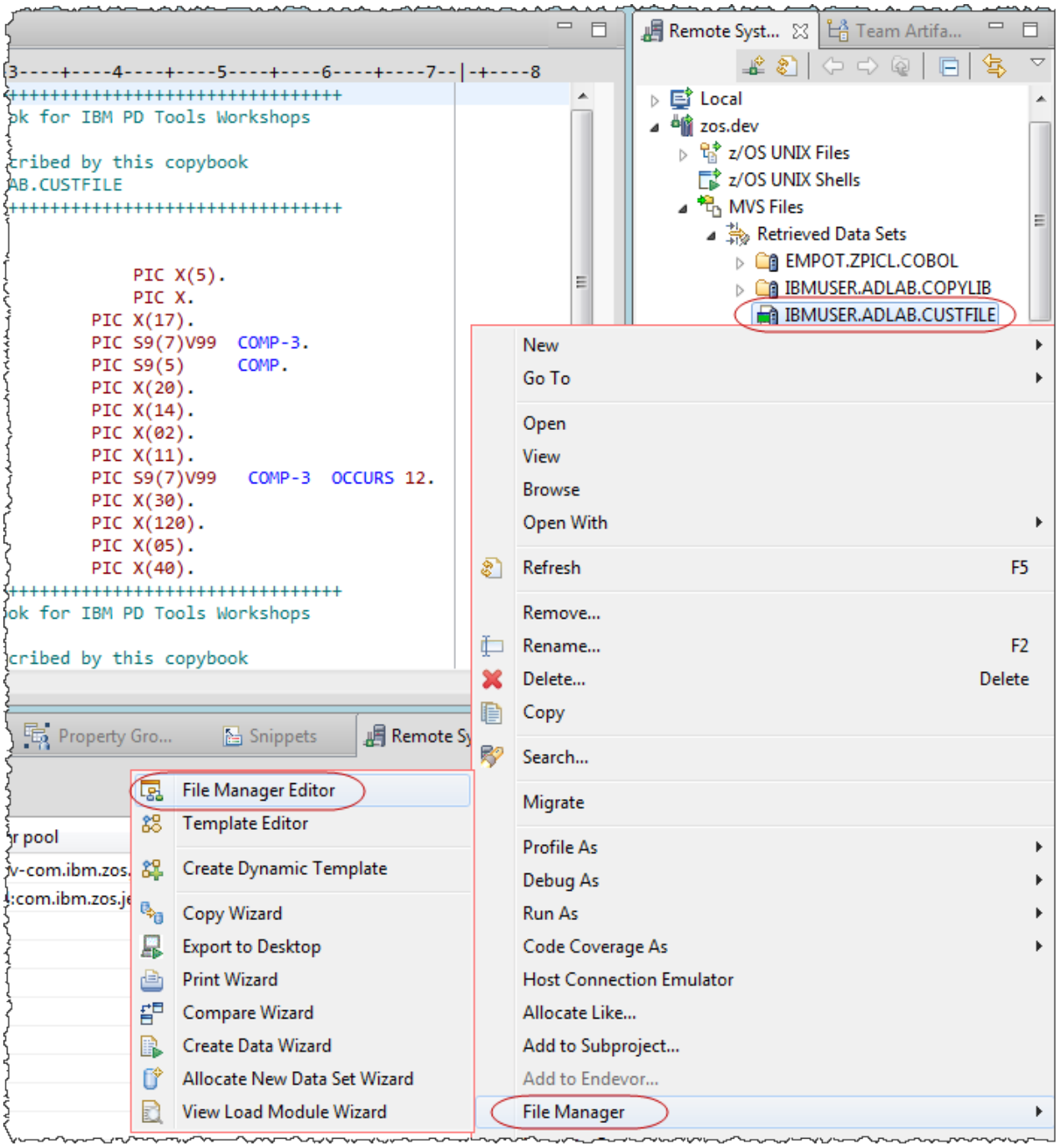
8.4.6 ⏭ Using the column **Selected** double-click on the boxes
1 **CUST-ID,** 2 **NAME** and 3 **ACCT-BALANCE**.
⏭ Click **OK.**

The template is created. At any time, you can edit and modify it.

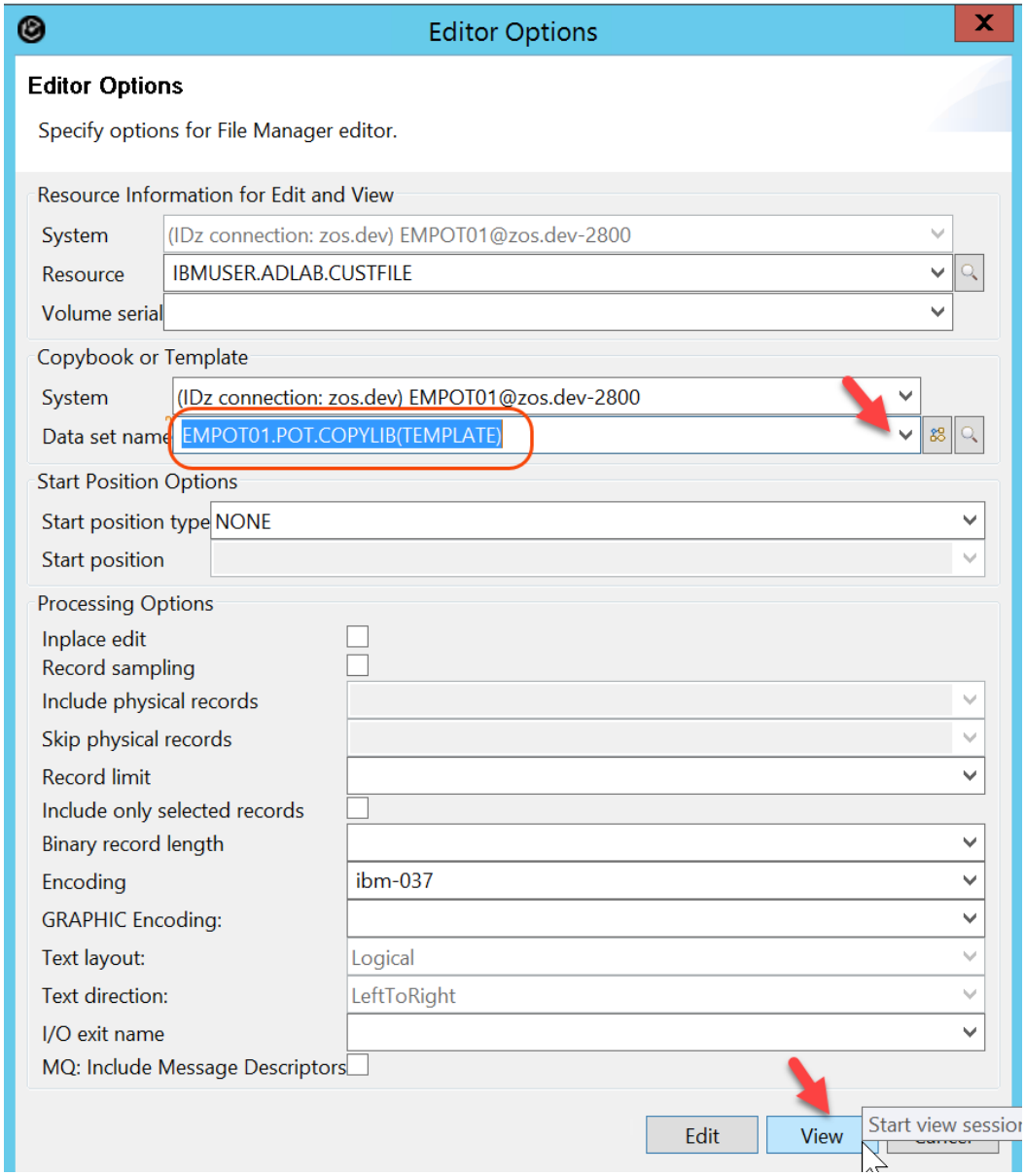


8.4.7 Now that the template is created you can use against the data set.

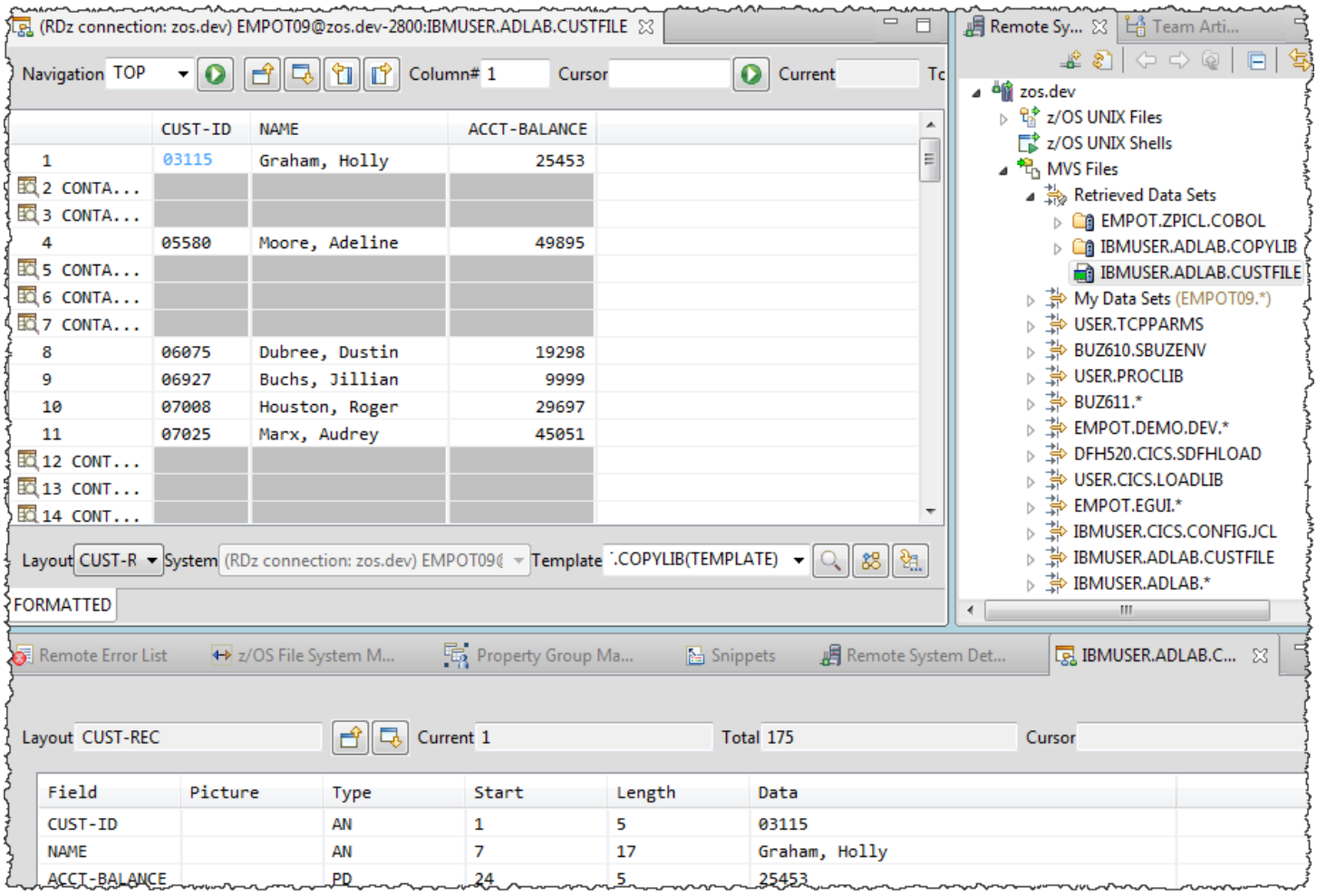
Right click on **IBMUSER.ADLAB.CUSTFILE** and select **File Manager -> File Manager Editor**.



8.4.8 On *Data set name* type the Template that you have created.
Type **EMPOT01.POT.COPYLIB(TEMPLATE)** (or use the drop down) and click **View**.



8.4.9 You should have the results below..
Notice that only 3 fields are displayed and all have account balance higher than **100.**



File manager had much more capabilities. But at least you had an idea how it works with z/OS datasets

8.4.10 Use **Ctrl + Shift + F4** to close all editors. Or just click on the . of each opened editor

Congratulations! You have completed the Lab 1.

LAB 2A – (OPTIONAL) Using IBM zUnit to Unit Test a COBOL CICS/DB2 program using Remote assets (60 minutes)

Updated May 28, 2021 by Regi – Created by Regi/Wilbert

Acknowledgments:
We would like to thank the following for their assistance:
Suman Gopinath & Nathan Cassata.

This lab will take you through the steps of using the automated unit testing (zUnit) capabilities of IBM Developer for z (IDz) to create a unit test case for a COBOL CICS/DB2 program. This enables the testing of just a single program within a CICS transaction without the need to run the whole transaction. This is done by stubbing out CICS calls, enabling the program to be tested without a CICS environment and without the need to deploy to CICS after a code change. This enables a developer to test early without impacting other developers that share the same CICS environment.

In this lab you will record interaction with a COBOL CICS/DB2 program (program under test) and import the recorded data into the unit test case generation process. You will build and run the unit test, make a change to the CICS COBOL program, and rerun the unit test.

Overview of development tasks

To complete this tutorial, you will perform the following tasks:

1. **Get familiar with the application using the 3270 terminal**
   → You will start a 3270 emulation and execute a transaction named **HCAZ** to become familiar with the Application that you will be recording.
2. **Import a z/OS project**
   → You will import a z/OS project with the required resources added to the project.
3. **Record interaction with the application.**
   → You will record an interaction with the COBOL CICS/DB2 program.
4. **Generate, build and run the unit test**
   → You will compile and link-edit the generated unit test program, followed by running the unit test.
5. **Modify the program and rerun the unit test**
   → You will modify the program under test, rerun the unit test, and observe the failure of the test case.
6. **Run the unit test from a batch JCL .**
   → You will run the unit test from a Batch JCL and observe a similar test case result.

Section 1. Get familiar with the application using the 3270 terminal

You will start a 3270 emulation and execute a transaction named **HCAZ** to become familiar with the Application that you will be interacting with.

___ 1.1 Connect to z/OS and emulate a CICS 3270 terminal

You will use IDz to emulate a 3270 terminal to run the CICS transaction.

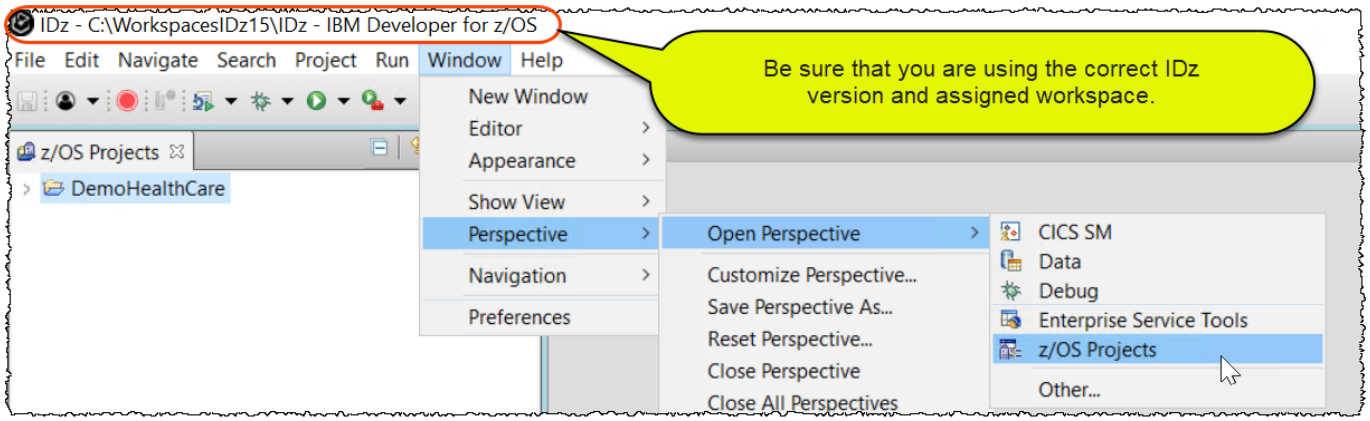
1.1.1 Start *IBM Developer for z Systems version 15* if it is not already started

⏭ Using the desktop double click on **IDz V15** icon.

⏭ Verify that the message indicates that it is Version **15.0.1**

IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon.

1.1.2 ⏭ Open the **z/OS Projects** perspective by selecting
Window > Perspective > Open Perspective > z/OS Projects



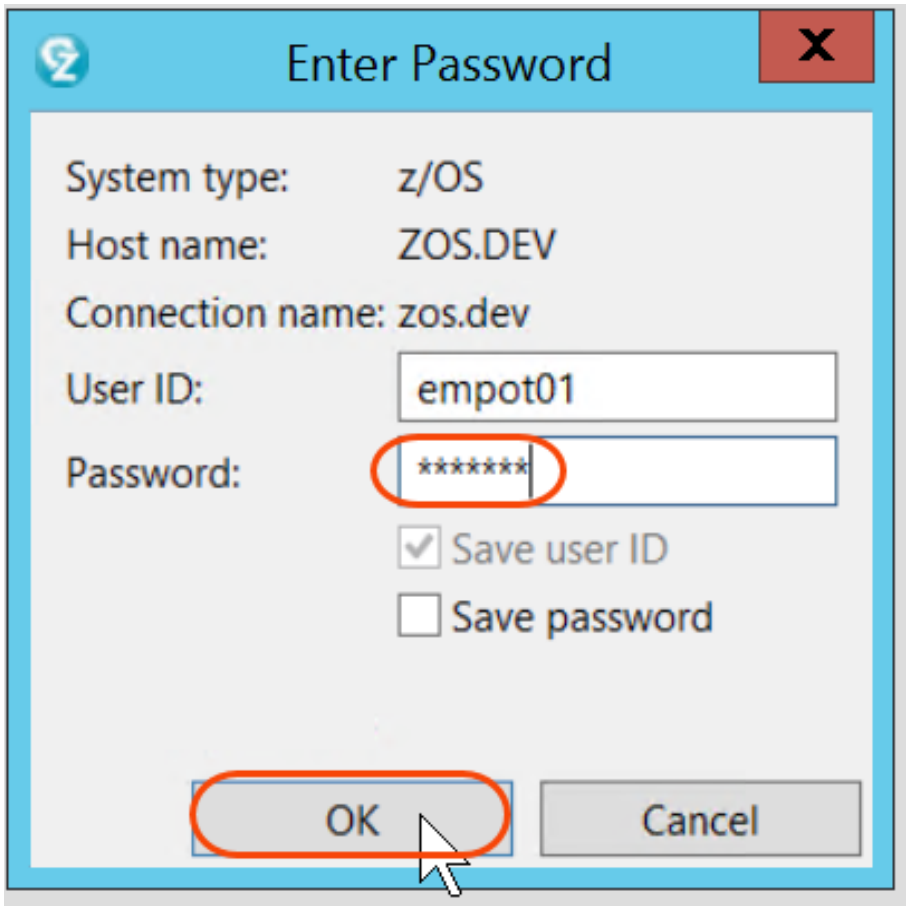
1.1.3 You will use userid **empot01** .

If you are connected as empot01, jump to step 1.1.5 Otherwise.

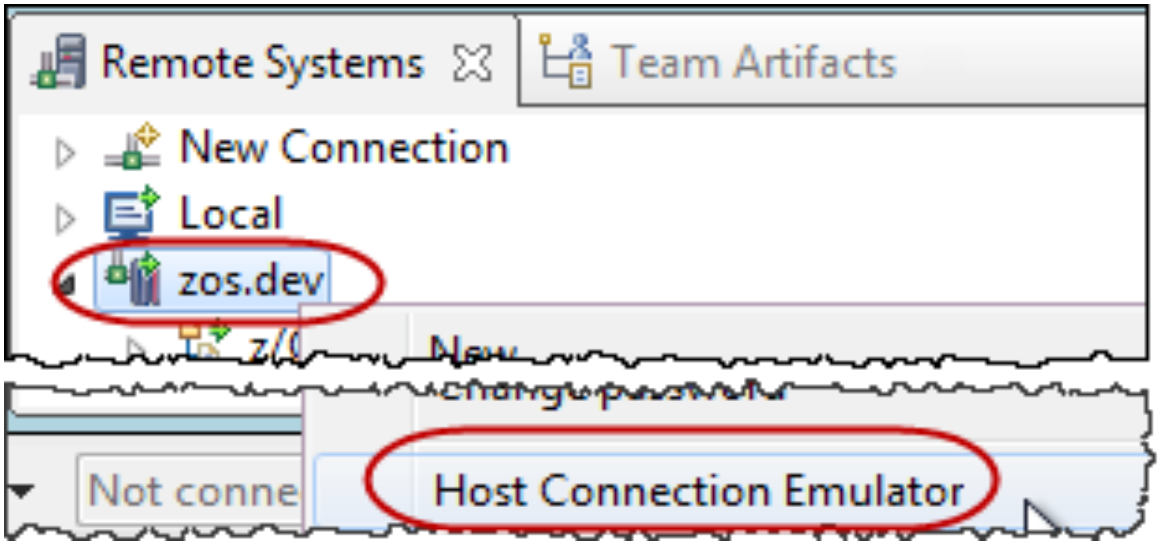
⏭ Using *Remote Systems* view, right click on **zos.dev** and select **Connect**

(You also may need to disconnect first)

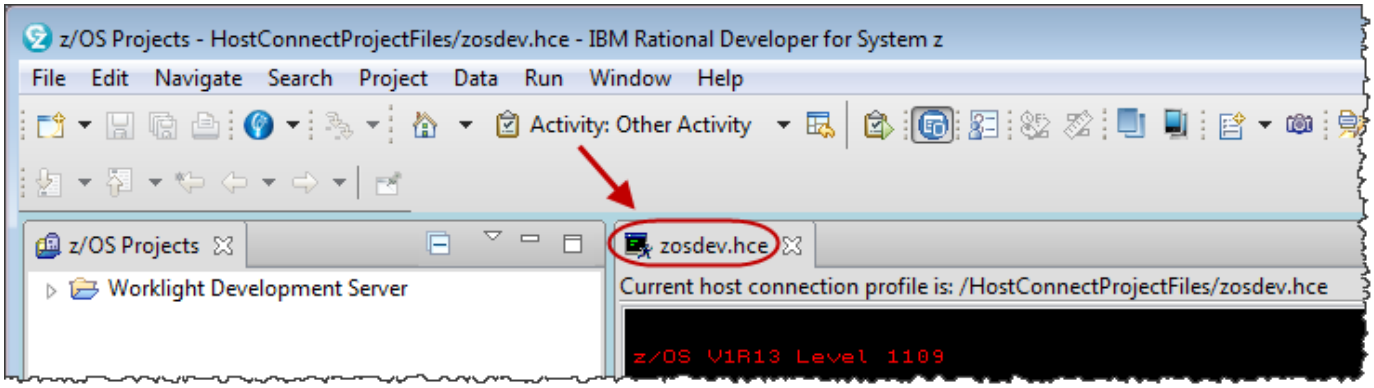
1.1.4 Type **empot01**as userid and **empot01** as password.
The userid and password can be any case; don't worry about having it in UPPER case.
Click **OK** to connect to z/OS.



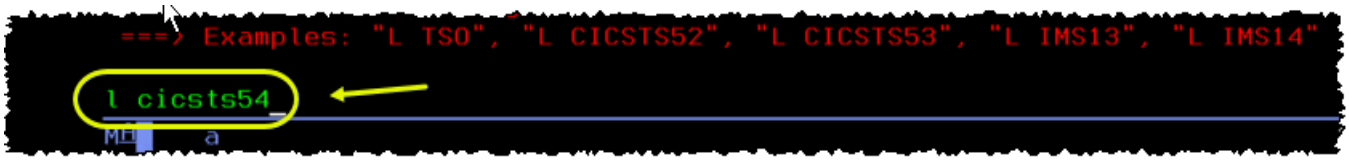
1.1.5 Using the *Remote Systems* view, right click on **zos.dev** and select **Host Connection Emulator.**



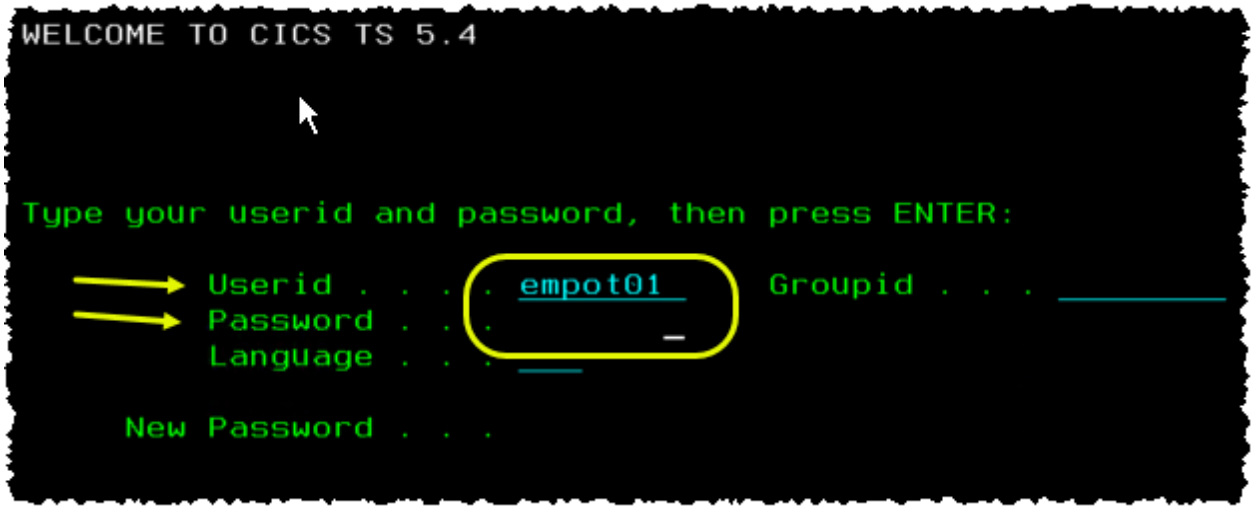
1.1.6 Since you will need more space, **double-click** on the ***zos.dev.hce*** title



1.1.7 Type **l cicsts54**. (where “l” is the lower case of letter “L”) and press **Enter key**.



1.1.8 Logon using your z/OS user id and password (**empot01)** and press **Enter**.

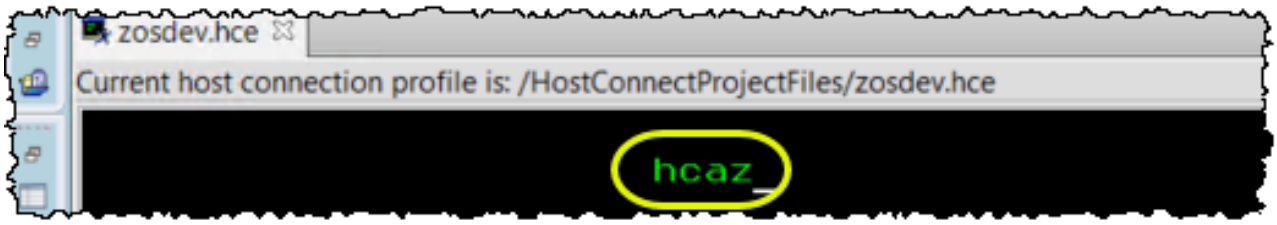


1.1.9 The sign-on message is displayed

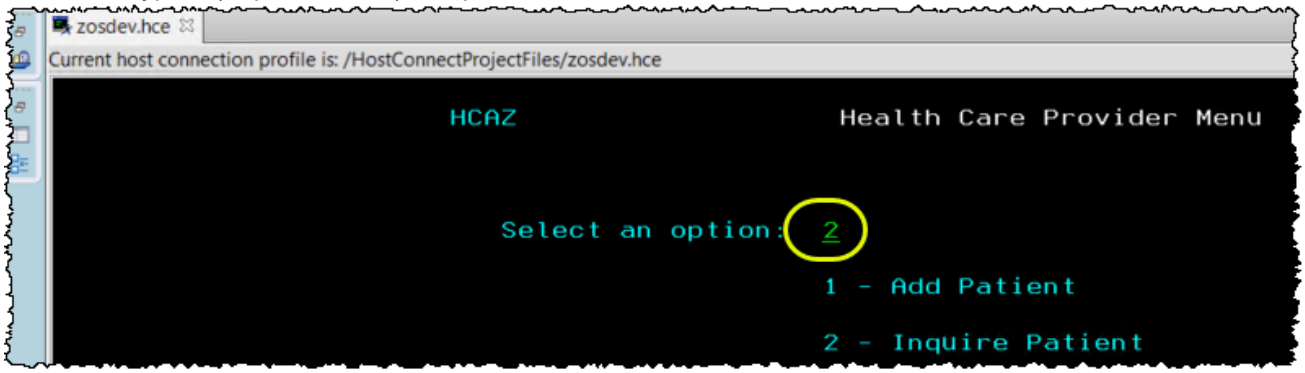
___ 1.2 Run CICS transaction HCAZ

You should now be in the z/OS CICS region named *CICSTS54*. This is the CICS instance where you will make the recording of your interaction.

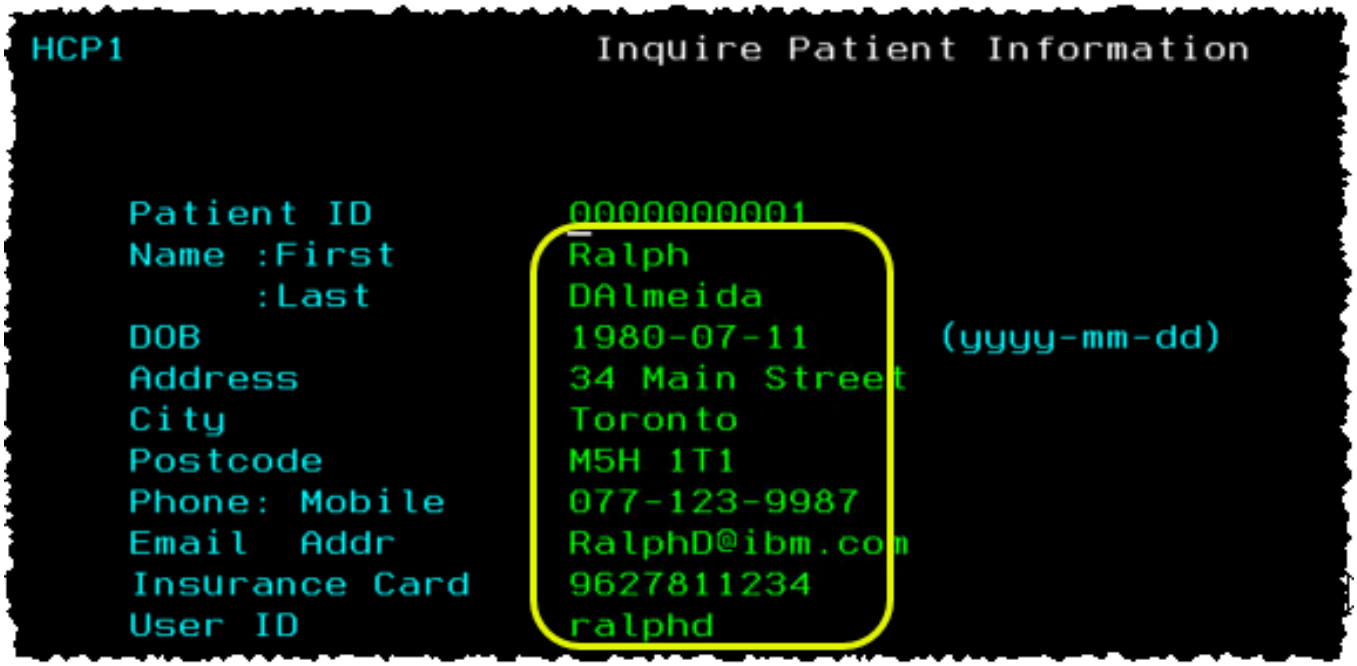
1.2.1 Type the CICS transaction **hcaz** and press the **Enter** key.



1.2.2 Type **2** (Inquire Patient) and press **Enter** .

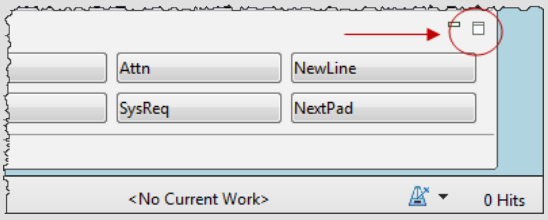


1.2.3 Type **1** for Patient ID and press Enter. The program will read the patient from a DB2 table and display the customer details. Remember the patient first name: **Ralph**

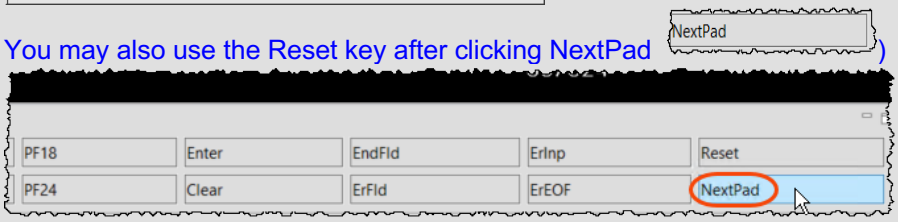


IF you need to use functions like Clear or Reset

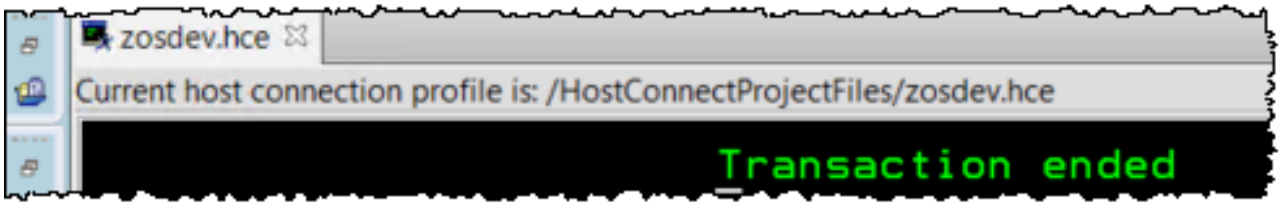
*If you need to use the **clear** function use the key **Esc**.*
.Also you may look in the right lower corner, select this icon . This will display possible keys, including the clear button.



You may also use the Reset key after clicking NextPad)

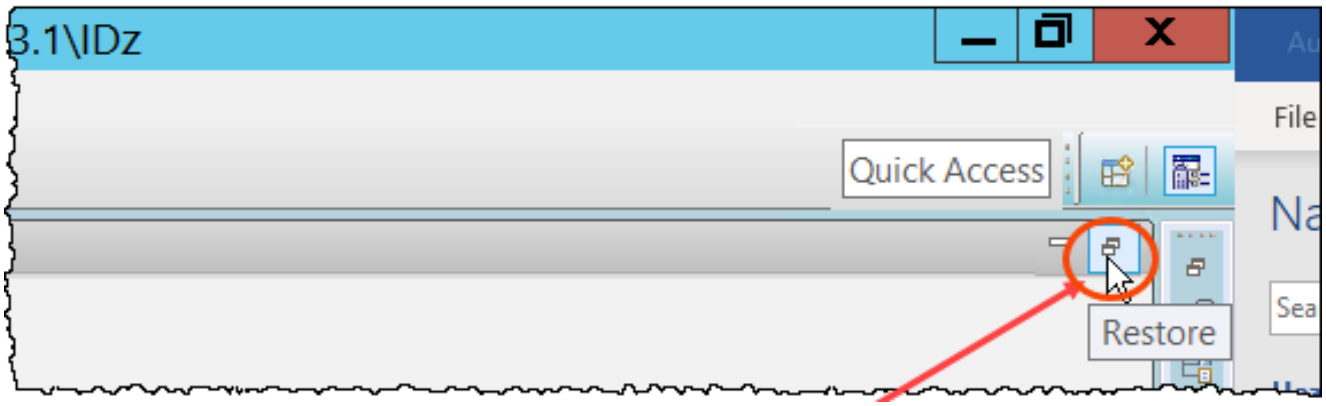


1.2.4 Press **F3** to end the application.



1.2.5 **Close** the terminal emulation clicking on → . Or pressing **CTRL + Shift + F4**.

1.2.6 You may need to restore the **z/OS Projects** perspective by clicking on the icon on top right:



What have you done so far?
You emulated a 3270 terminal using IBM Developer for System z.
You also executed the CICS transaction **HCAZ** and verified a simple interaction with the Hospital application. The objective here was to show the code that you will update.

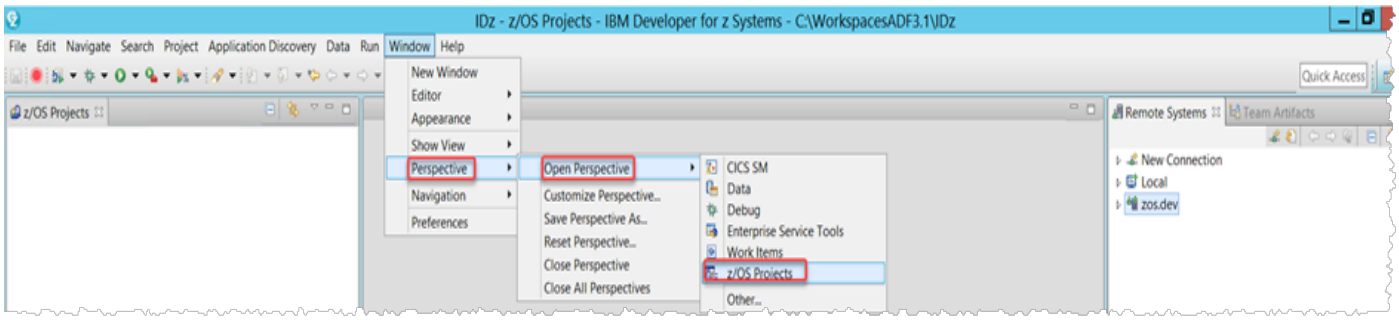
Section 2 – Import a z/OS project

You will import a z/OS Project that contains the resources needed to generate a unit test program for a COBOL CICS application.

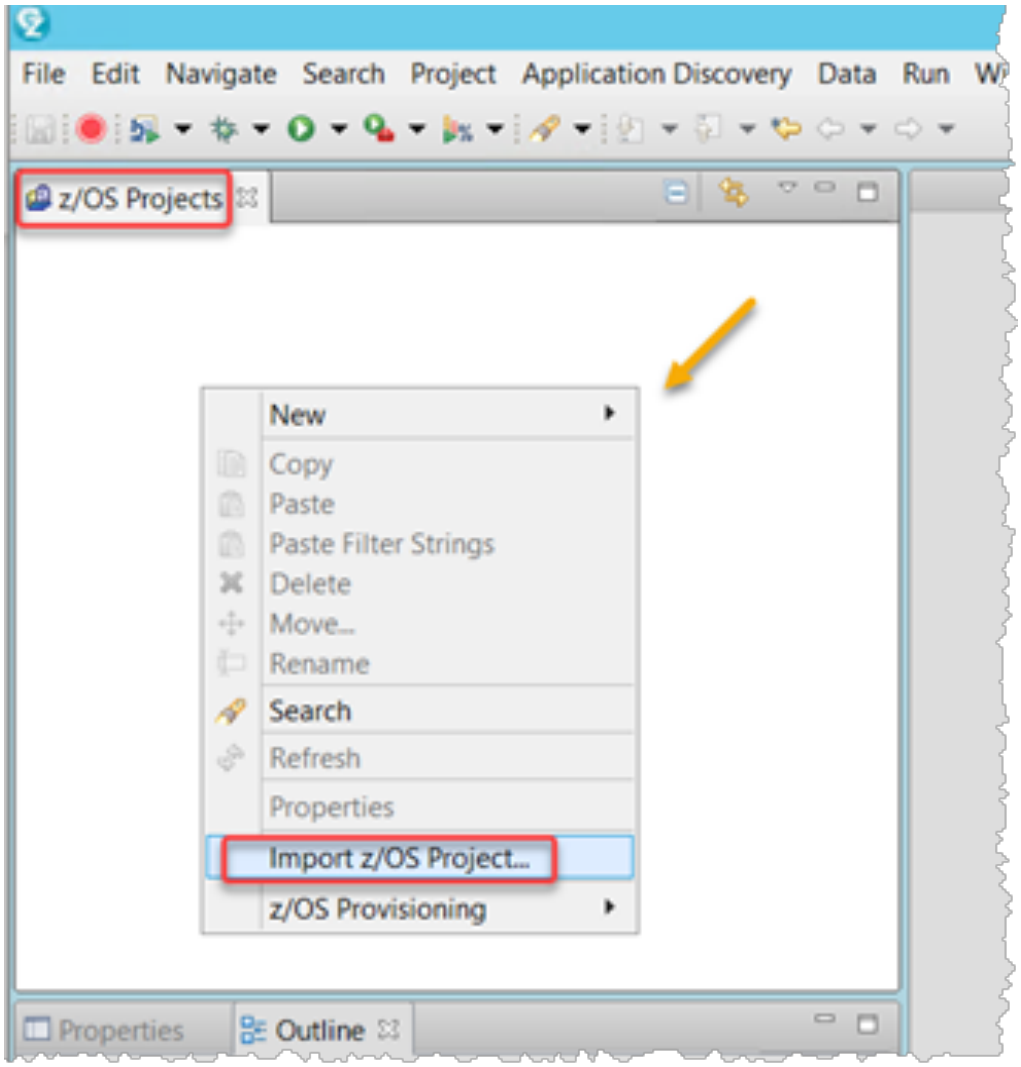
___ 2.1 Importing the LAB6B z/OS Project

The Hospital Application used in this lab has its source code in a *PDS* that's been added to a z/OS Project. You must be connected to the zos.dev system (see step 1.1).

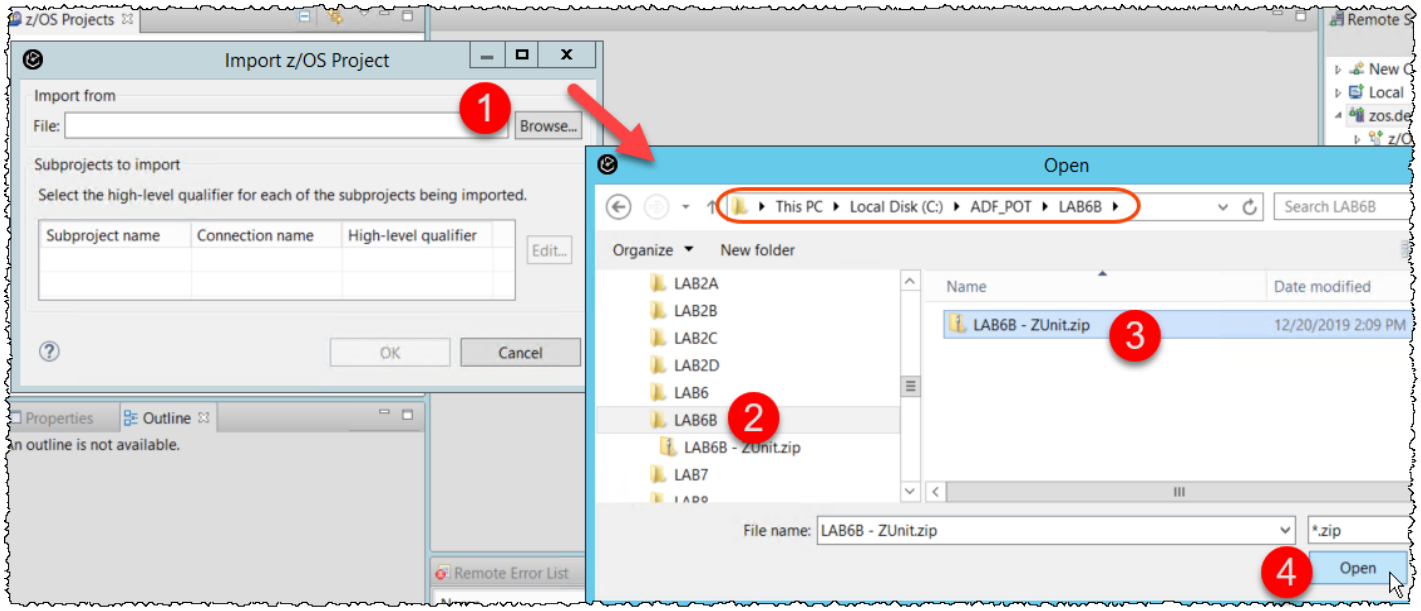
2.1.1 If not already in the z/OS Projects perspective, open the **z/OS Projects** perspective by selecting **Window > Perspective > Open Perspective > z/OS Projects**



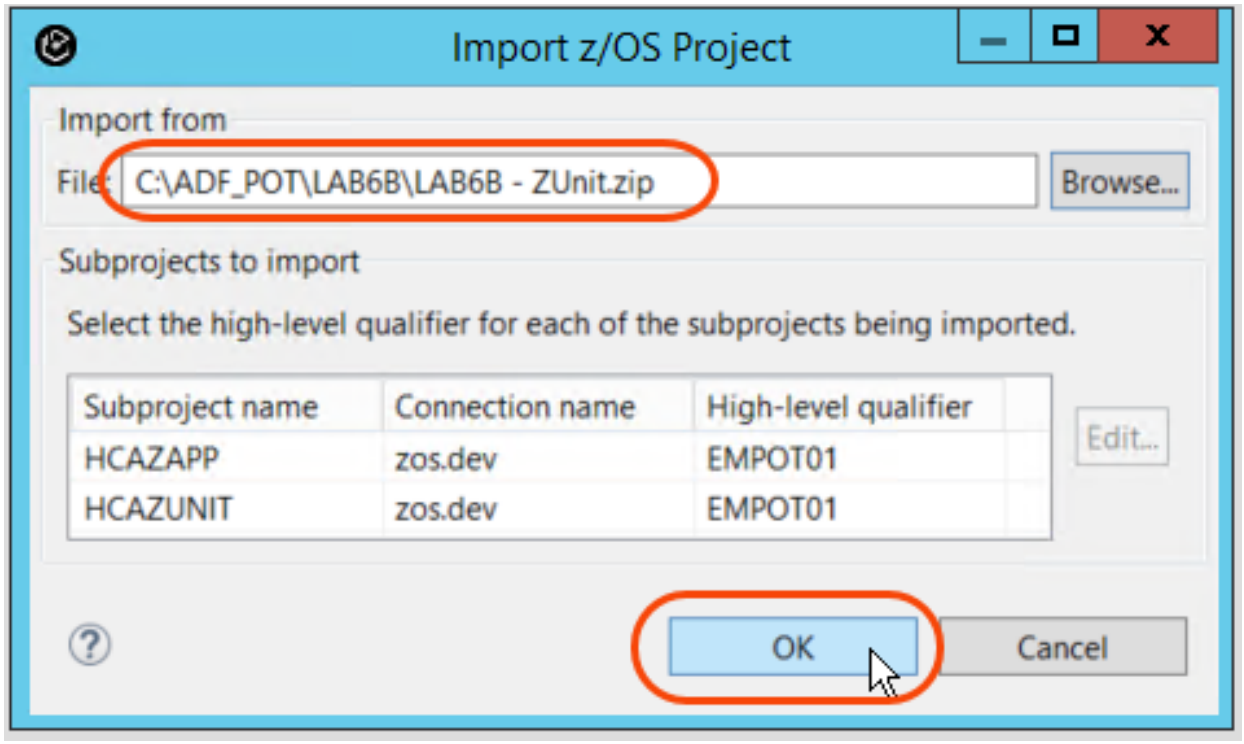
2.1.2 Using the **z/OS Projects** view on top left, position the cursor anywhere in the view. Right mouse click and select the **Import z/OS Project** action.


File Edit Navigate Search Project Application Discovery Data Run
z/OS Projects
New
Copy
Paste
Paste Filter Strings
Delete
Move...
Rename
Search
Refresh
Properties
Import z/OS Project...
z/OS Provisioning
Properties
Outline

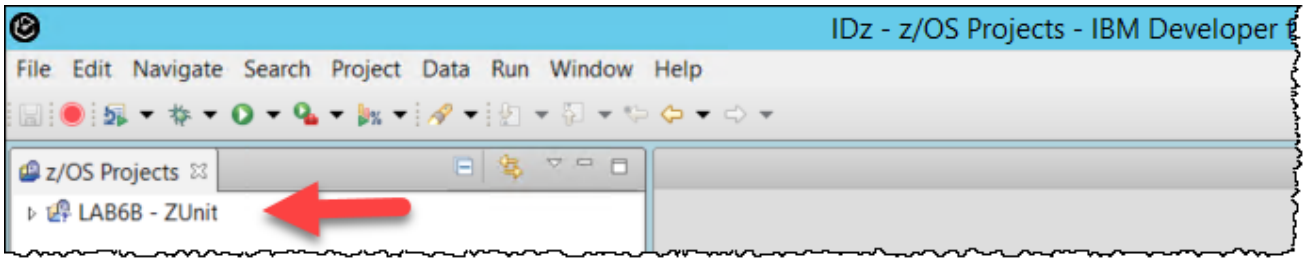
2.1.3 In the **Import z/OS Project** dialog, click the **Browse** button, navigate to **`C:\ADF_POT\LAB6B`** and select **LAB6B - ZUnit.zip**. Click **Open**.



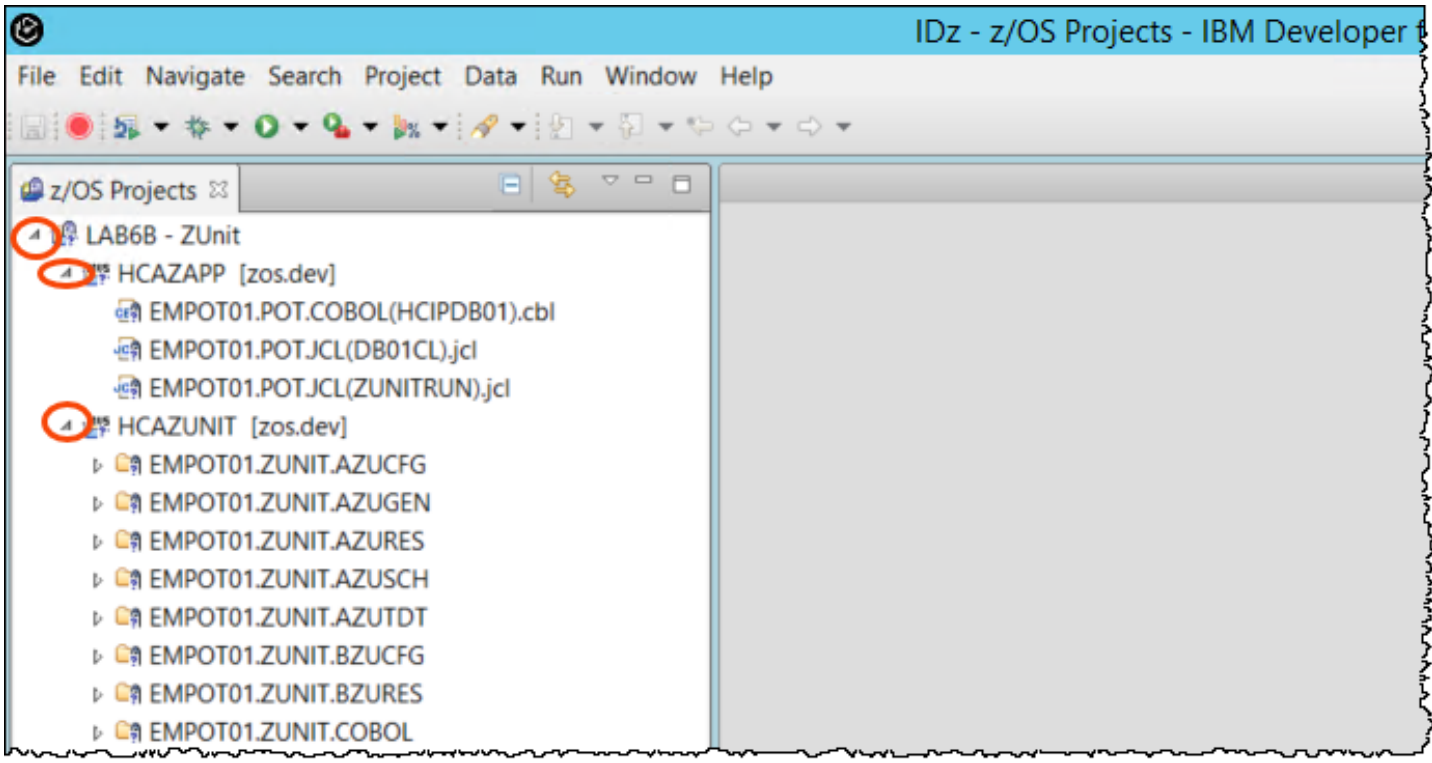
2.1.4 In the dialog that opens, click **OK**.



You should see something like this in your z/OS Projects view:


IDz - z/OS Projects - IBM Developer
File Edit Navigate Search Project Data Run Window Help
z/OS Projects
LAB6B - ZUnit

2.1.5 **Expand the nodes** by left clicking on the icon as shown below:.



Section 3 – Record data interaction using the CICS application.

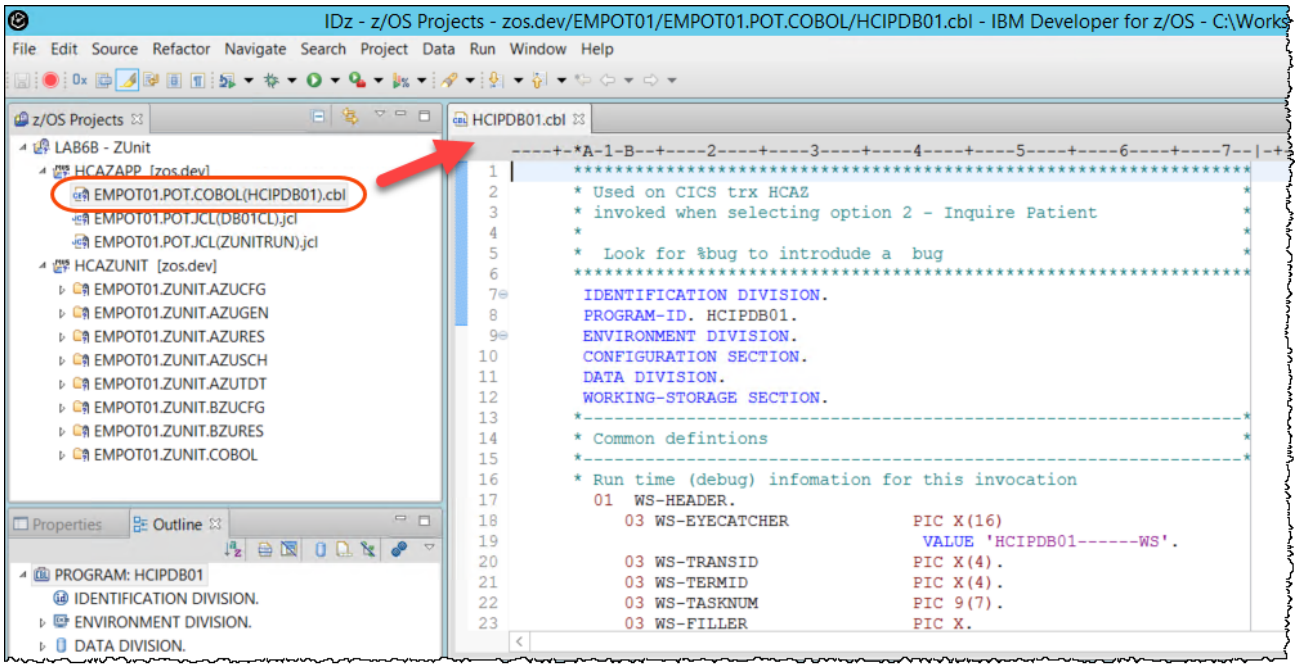
Using IDz you will record the interaction with the COBOL/DB2 program that reads a patient from a DB2 table.

___ 3.1 Understanding the COBOL program that reads from DB2 table

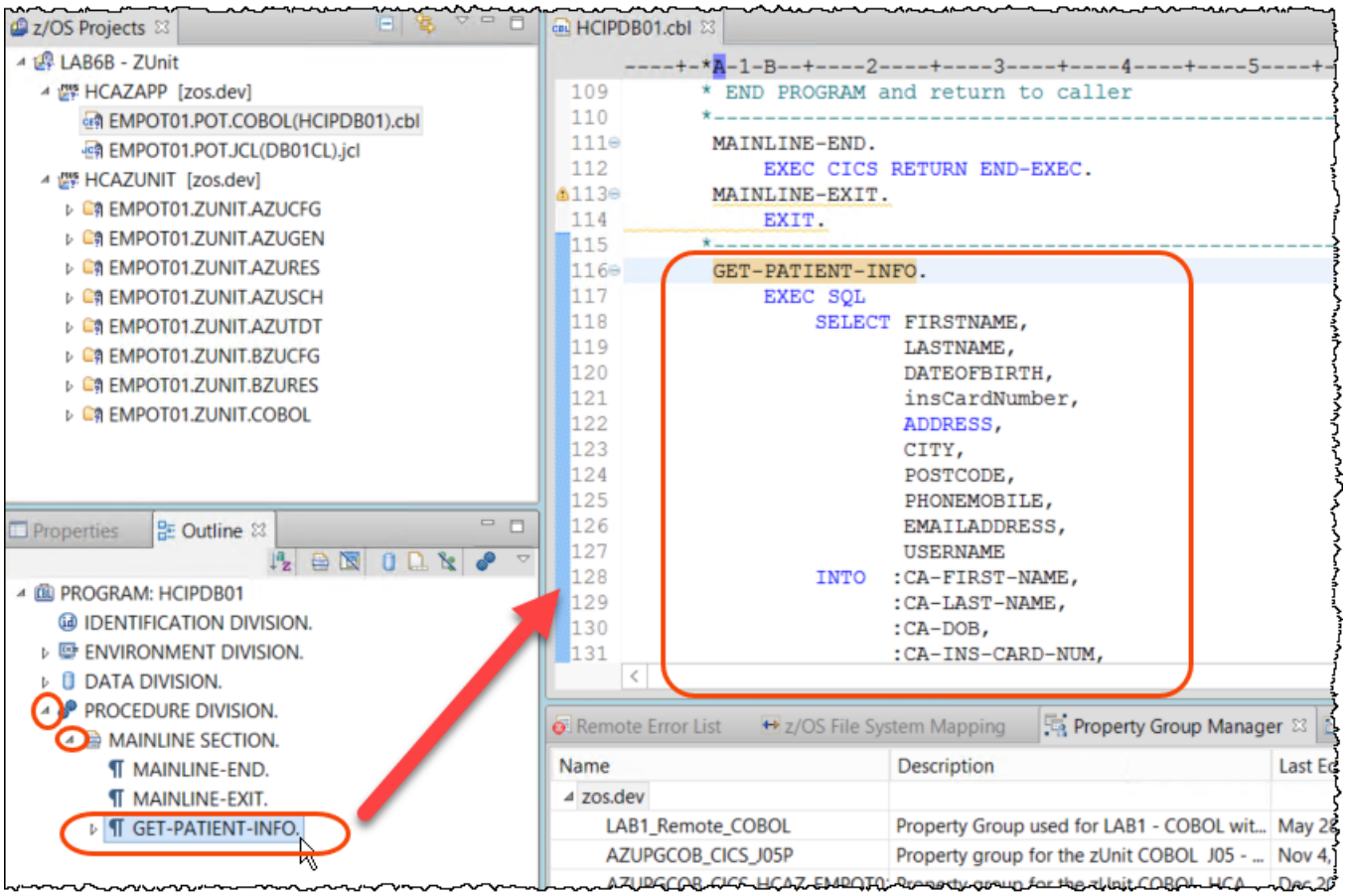
The COBOL code that reads the patient from the DB2 tables is the program ***HCIPDB01***

3.1.1 Using *z/OS Projects* view double click **EMPOT01.POT.COBOL(HCIPDB01).cbl** under **HCAZAPP**.

This is the program that you will update later on.

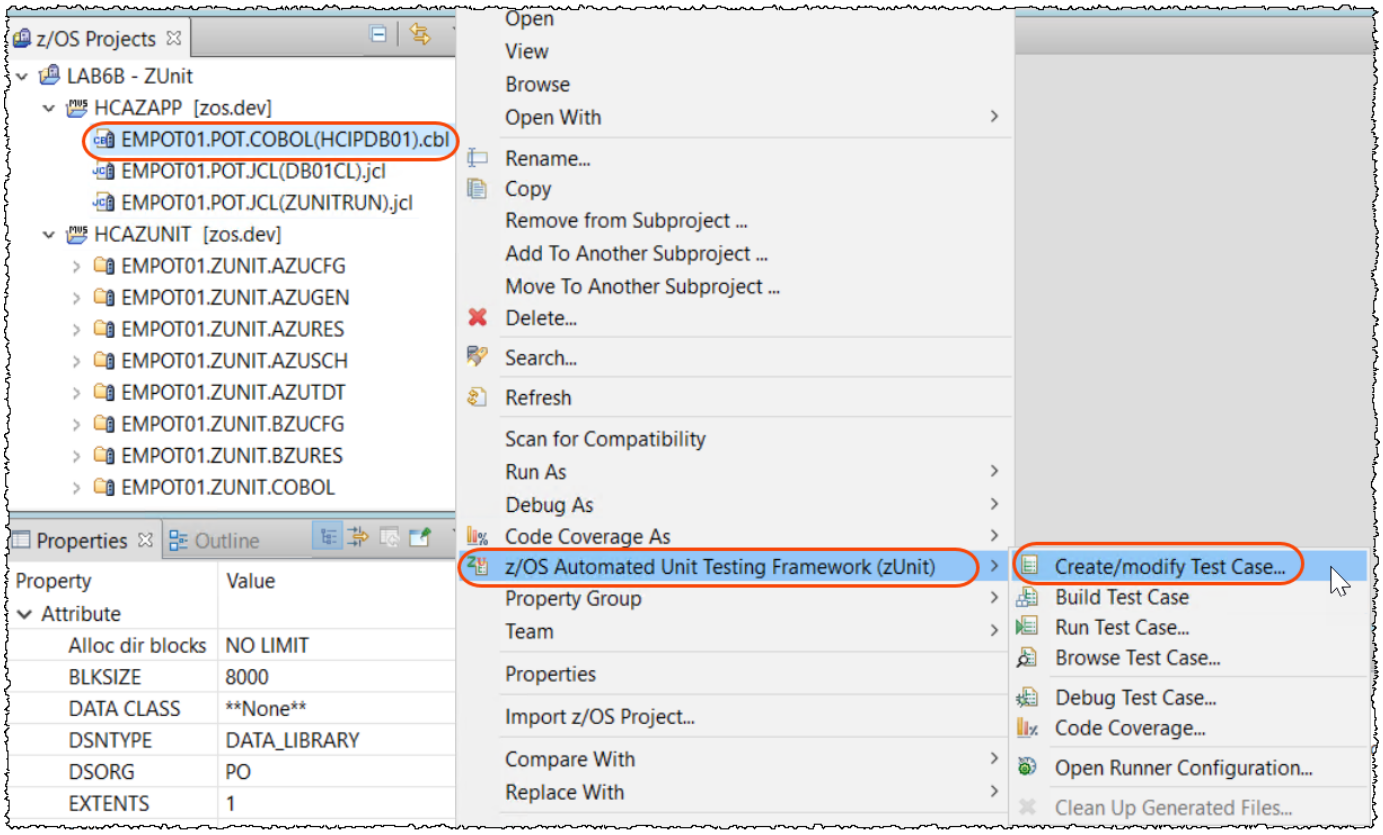

IDz - z/OS Projects - zos.dev/EMPOT01/EMPOT01.POT.COBOL/HCIPDB01.cbl - IBM Developer for z/OS - C:\Works
File Edit Source Refactor Navigate Search Project Data Run Window Help
z/OS Projects
LAB6B - ZUnit
HCAZAPP [zos.dev]
EMPOT01.POT.COBOL(HCIPDB01).cbl
EMPOT01.POT.JCL(DB01CL).jcl
EMPOT01.POT.JCL(ZUNITRUN).jcl
HCAZUNIT [zos.dev]
EMPOT01.ZUNIT.AZUCFG
EMPOT01.ZUNIT.AZUGEN
EMPOT01.ZUNIT.AZURES
EMPOT01.ZUNIT.AZUSCH
EMPOT01.ZUNIT.AZUTDT
EMPOT01.ZUNIT.BZUCFG
EMPOT01.ZUNIT.BZURES
EMPOT01.ZUNIT.COBOL
Properties
Outline
PROGRAM: HCIPDB01
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
HCIPDB01.cbl
1 * Used on CICS trx HCAZ
2 * invoked when selecting option 2 - Inquire Patient
3 *
4 * Look for %bug to introdude a bug
5 IDENTIFICATION DIVISION.
6 PROGRAM-ID. HCIPDB01.
7 ENVIRONMENT DIVISION.
8 CONFIGURATION SECTION.
9 DATA DIVISION.
10 WORKING-STORAGE SECTION.
11 * Common defintions
12 * Run time (debug) infomation for this invocation
13 01 WS-HEADER.
14 03 WS-EYECATCHER PIC X(16)
15 VALUE 'HCIPDB01------WS'.
16 03 WS-TRANSID PIC X(4).
17 03 WS-TERMID PIC X(4).
18 03 WS-TASKNUM PIC 9(7).
19 03 WS-FILLER PIC X.

3.1.2 Using the Outline view on left expand **PROCEDURE DIVISION** and click on **GET-PATIENT-INFO.** This is where the patient data is read from the DB2 table.
Later on you will introduce a bug in this program..

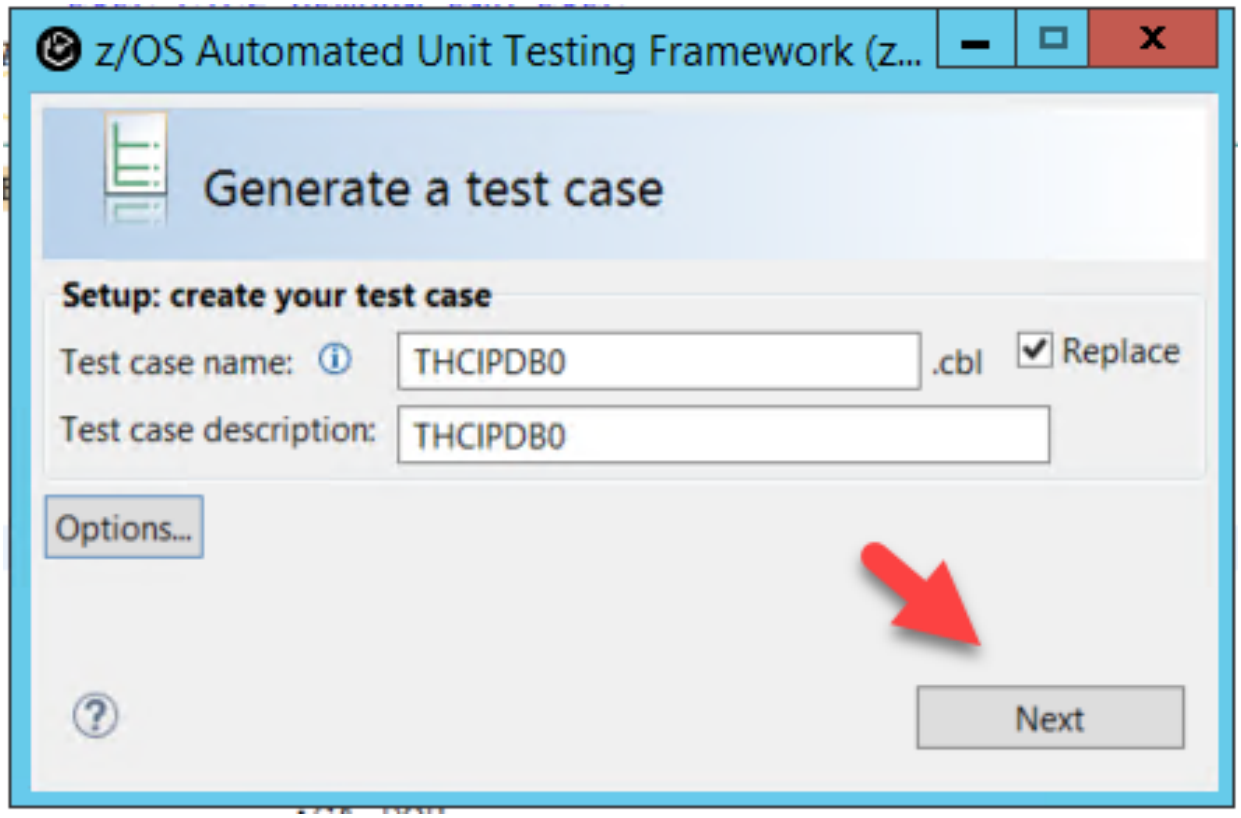


___ 3.2 Recording the COBOL program that sends the message

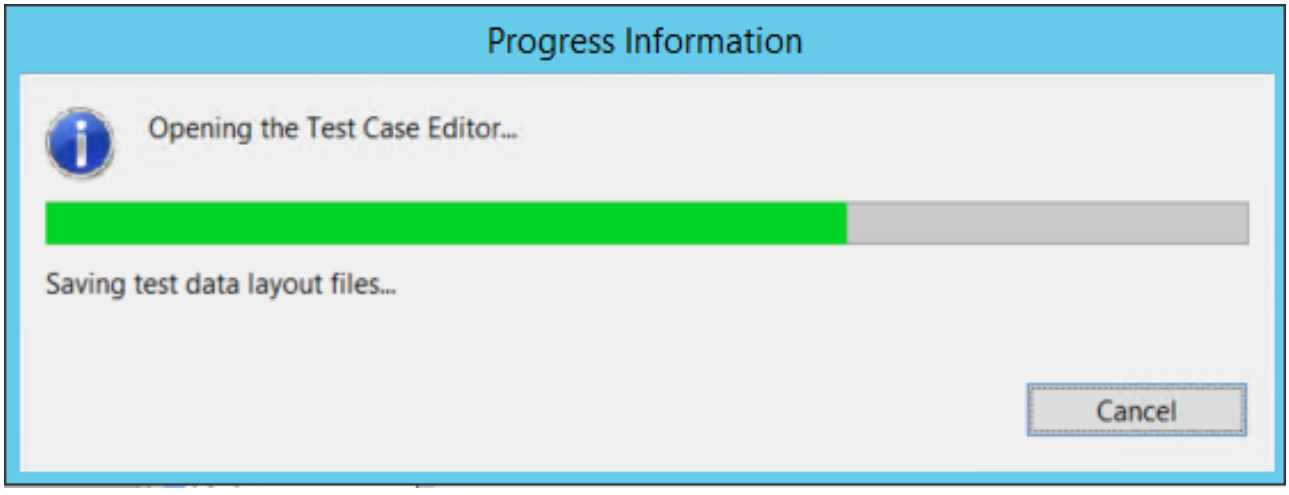
3.2.1 To start the recording, right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl** and select **z/OS Automated Unit Testing Framework (zUnit)-> Create/modify Test Case..**


z/OS Projects
LAB6B - ZUnit
HCAZAPP [zos.dev]
EMPOT01.POT.COBOL(HCIPDB01).cbl
EMPOT01.POT.JCL(DB01CL).jcl
EMPOT01.POT.JCL(ZUNITRUN).jcl
HCAZUNIT [zos.dev]
EMPOT01.ZUNIT.AZUCFG
EMPOT01.ZUNIT.AZUGEN
EMPOT01.ZUNIT.AZURES
EMPOT01.ZUNIT.AZUSCH
EMPOT01.ZUNIT.AZUTDT
EMPOT01.ZUNIT.BZUCFG
EMPOT01.ZUNIT.BZURES
EMPOT01.ZUNIT.COBOL
Properties
Outline
Property
Value
Attribute
Alloc dir blocks
NO LIMIT
BLKSIZE
8000
DATA CLASS
None
DSNTYPE
DATA_LIBRARY
DSORG
PO
EXTENTS
1
Open
View
Browse
Open With
Rename...
Copy
Remove from Subproject ...
Add To Another Subproject ...
Move To Another Subproject ...
Delete...
Search...
Refresh
Scan for Compatibility
Run As
Debug As
Code Coverage As
z/OS Automated Unit Testing Framework (zUnit)
Property Group
Team
Properties
Import z/OS Project...
Compare With
Replace With
Create/modify Test Case...
Build Test Case
Run Test Case...
Browse Test Case...
Debug Test Case...
Code Coverage...
Open Runner Configuration...
Clean Up Generated Files...

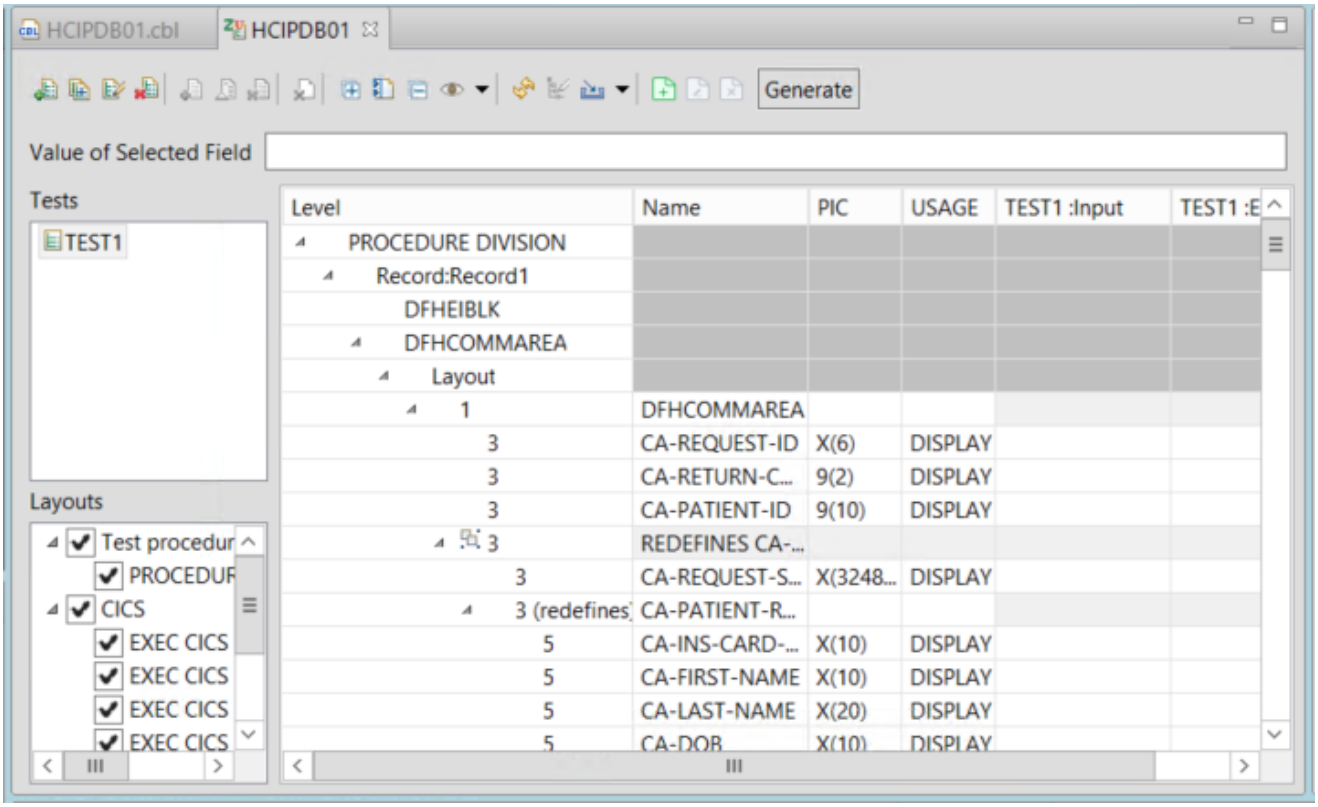
3.2.2 ⏭ This opens a dialog where you can name your test case.
Accept the auto-generated name and click **Next**.



3.2.3 This operation may take a while since members are being created on z/OS PDS **EMPOT01.ZUNIT.***
Be Patient!

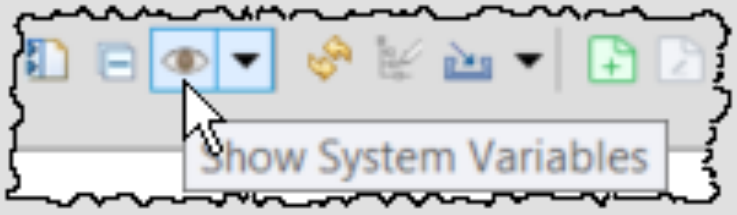


3.2.4 This will open the **Test Case Editor**, as shown below. TEST1 may or may not be on your screen.
If TEST1 is there you will delete on next step.



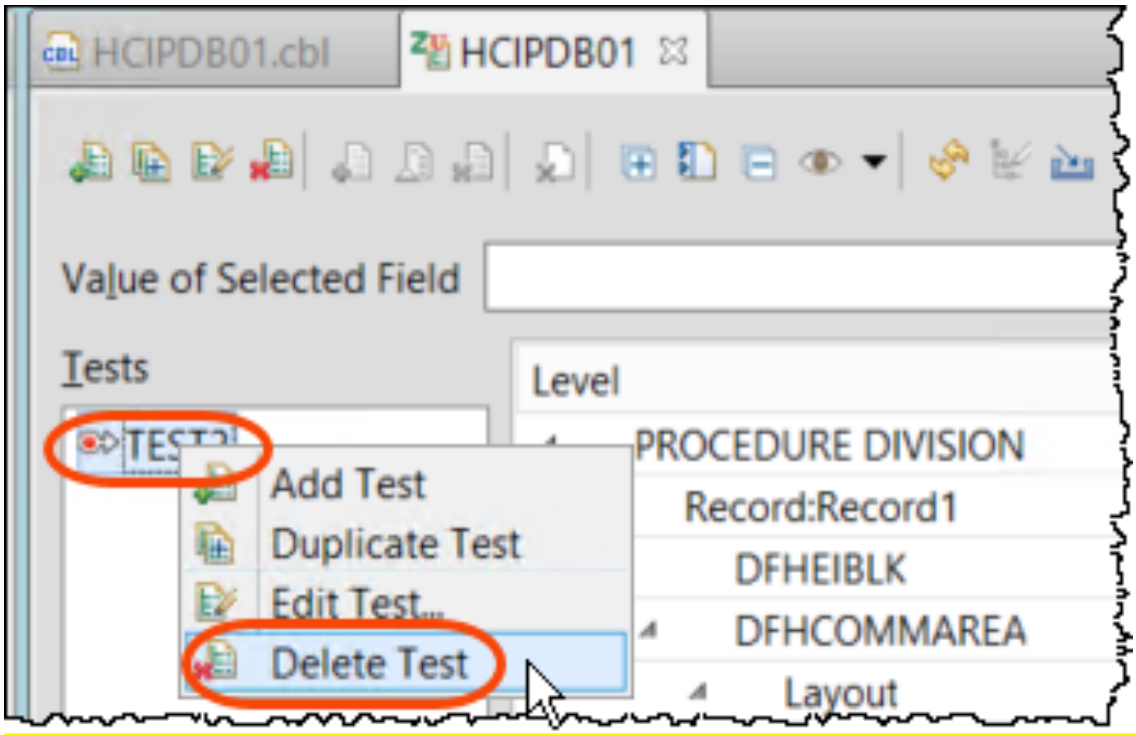
Understanding the test case editor

- The bottom left box summarizes all the input output variable structures – In this example, the program has a COMMAREA exchanged via the PROCEDURE DIVISION, five EXEC CICS statements and one EXEC SQL statement.
- The COMMAREA and the CICS statements are also accompanied by the CICS DFHEIBLK variables. To see those values you must select the icon



The variables that are returned by the program are the output from the program logic that a developer should be checking

3.2.5 Since we will be recording the test data, delete the **TESTx** entry (if it exists) by right clicking and selecting **Delete Test**

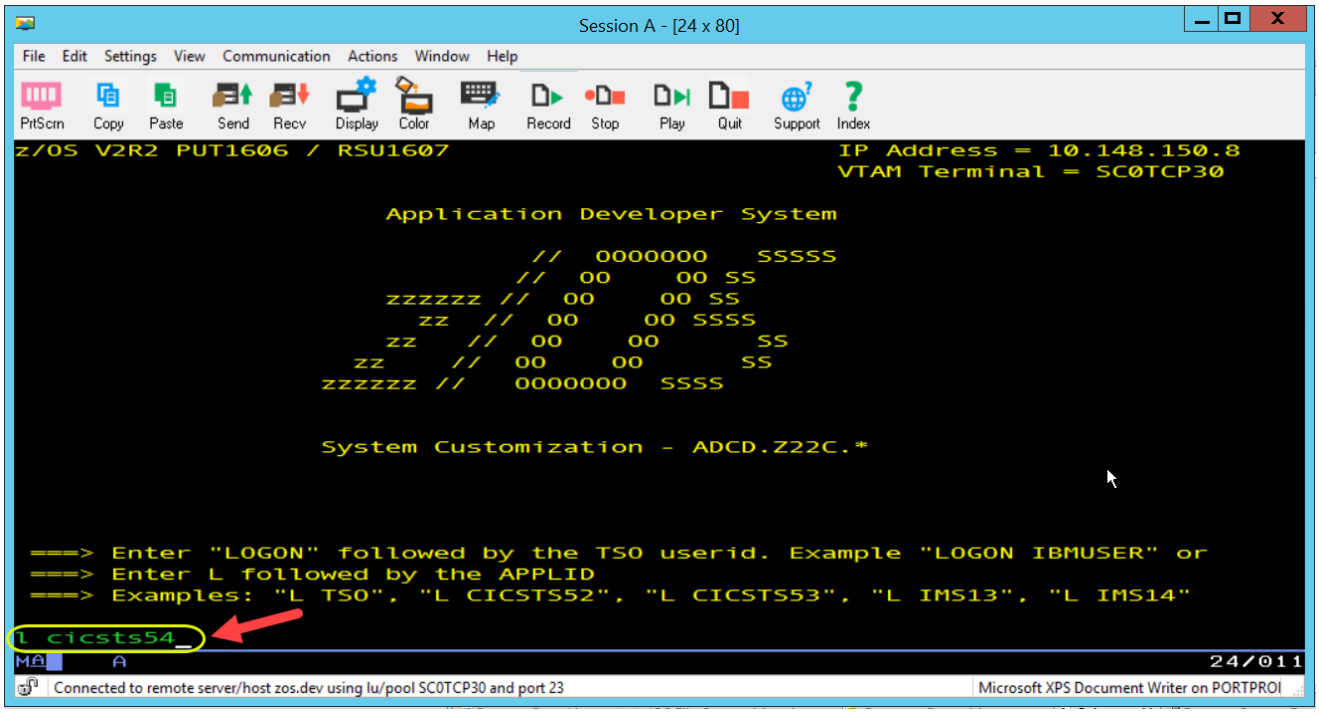


Notice that on section 1 we used the IDz emulator and now you will use the IBM PCOM emulator. Any 3270 emulator can be used.. We just show here a different way to emulate a 3270 terminal.

3.2.6 ⏭ Bring up a 3270 terminal emulator clicking on the **host emulator icon** on the Windows task bar.

This opens the host emulator.

3.2.7 ⏭ Type **l cicsts54**. (where l is L lower case) and press **Enter key** or the **right Ctrl** key.


Session A - [24 x 80]
File Edit Settings View Communication Actions Window Help
PrtScrn Copy Paste Send Recv Display Color Map Record Stop Play Quit Support Index
z/OS V2R2 PUT1606 / RSU1607
IP Address = 10.148.150.8
VTAM Terminal = SC0TCP30
Application Developer System
System Customization - ADCD.Z22C.*
===> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or
===> Enter L followed by the APPLID
===> Examples: "L TSO", "L CICSTS52", "L CICSTS53", "L IMS13", "L IMS14"
l cicsts54
24/011
Connected to remote server/host zos.dev using lu/pool SC0TCP30 and port 23
Microsoft XPS Document Writer on PORTPROI

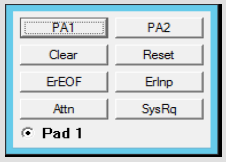
3.2.8 Sign on using **empot01** as the userid and **empot01** as the password.

3.2.9 Once you see the sign-on is complete message, enter **hcaz** and press the right **Ctrl** key.

IF you need to use functions like Clear or Reset

*If you need to use the **clear** or **reset** function right mouse click and select as below*

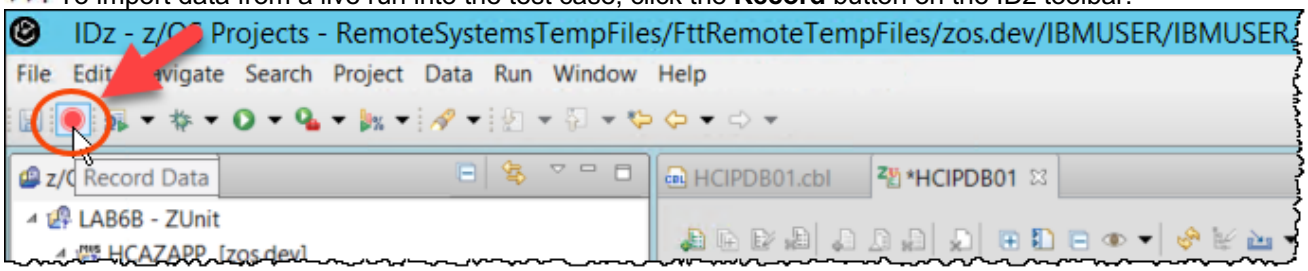
and



You are now ready to start recording and import data into the test case editor.

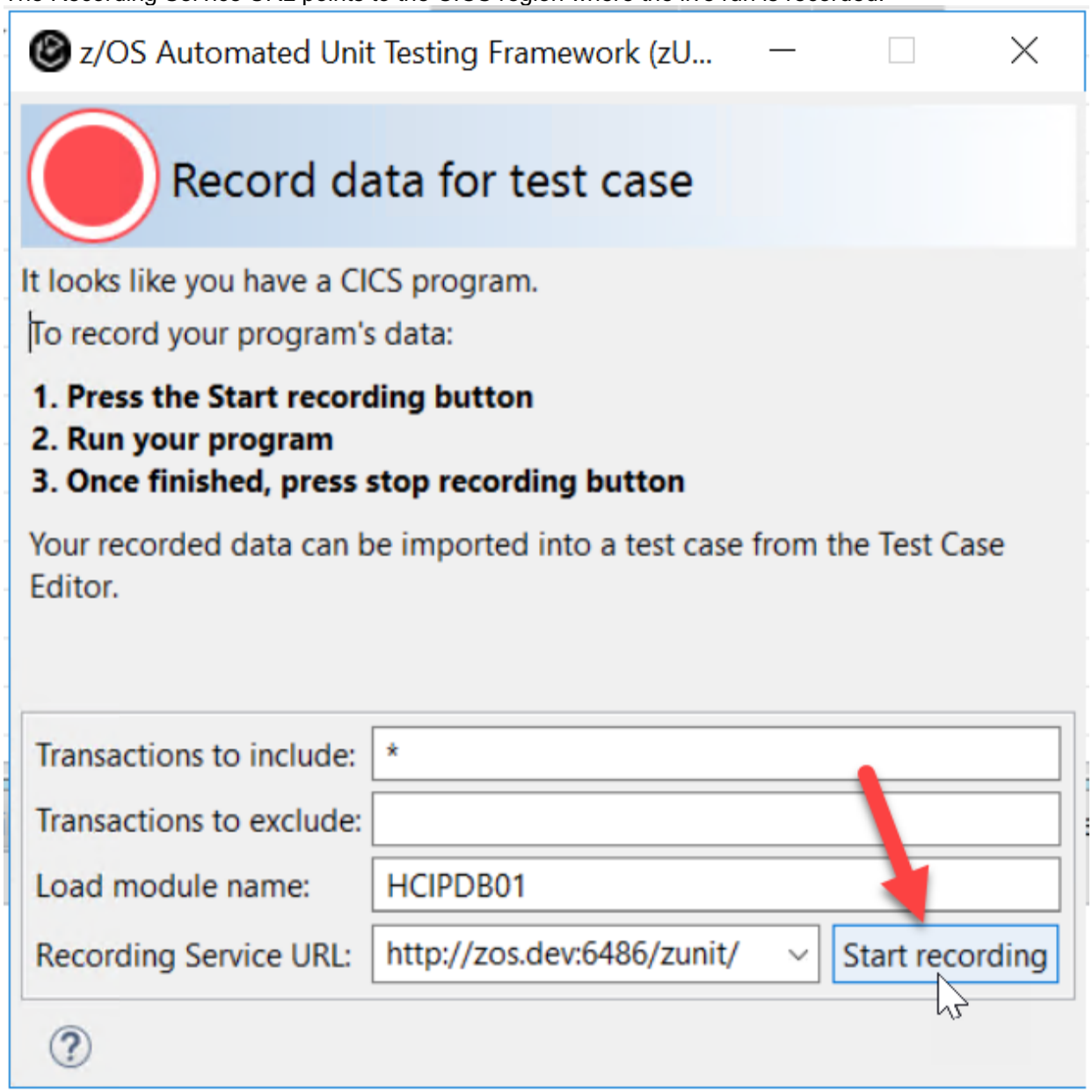
3.2.10 Minimize the terminal emulator and go back to IDz dialog.

To import data from a live run into the test case, click the **Record** button on the IDz toolbar.

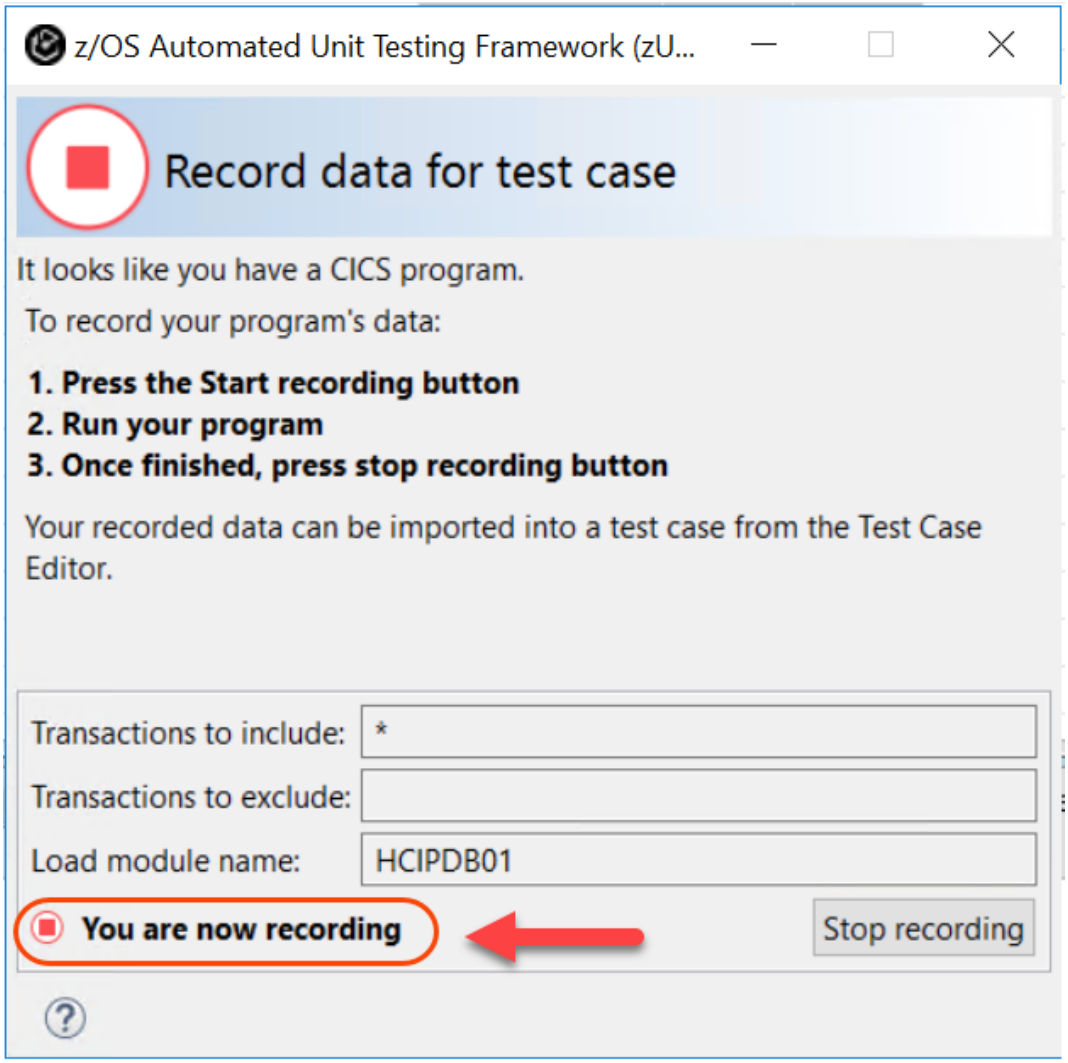


3.2.11 In the dialog that comes up, click on **Start recording**.

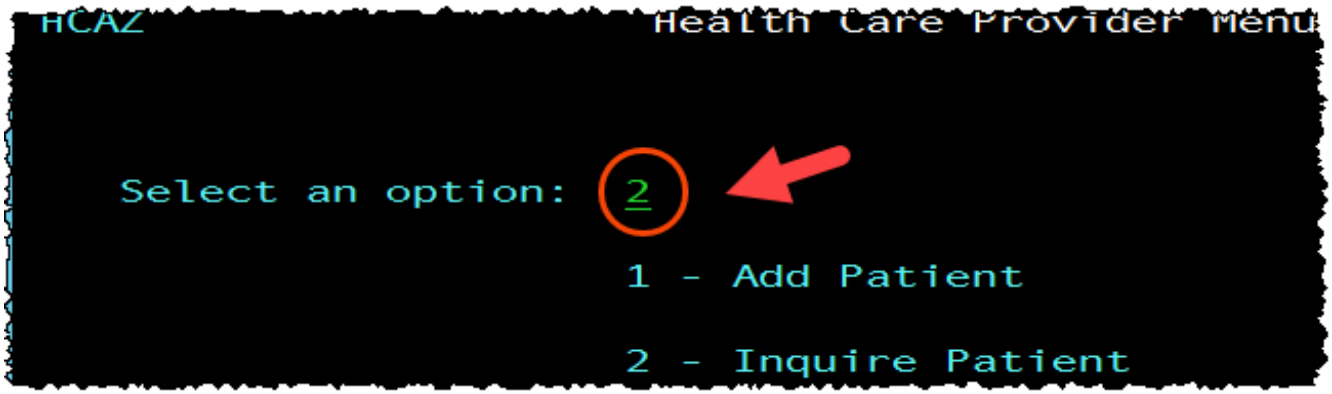
The Recording Service URL points to the CICS region where the live run is recorded.



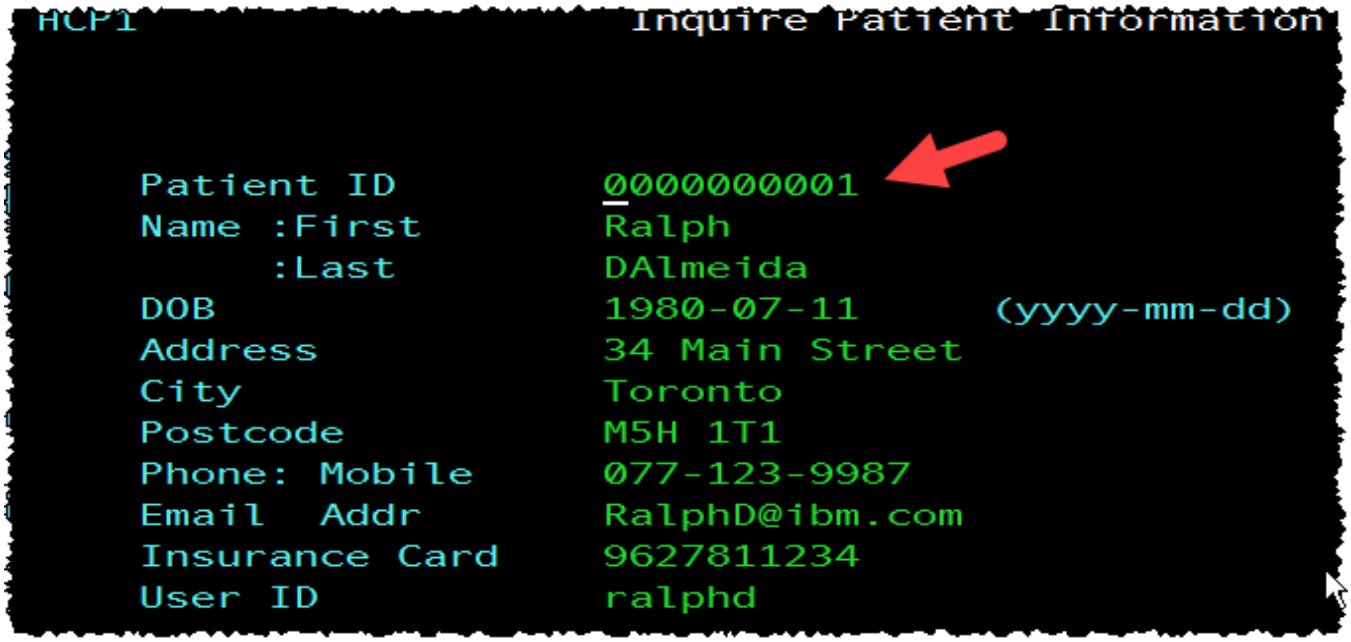
3.2.12 When recording is turned on, the message "**You are now recording**" appears.



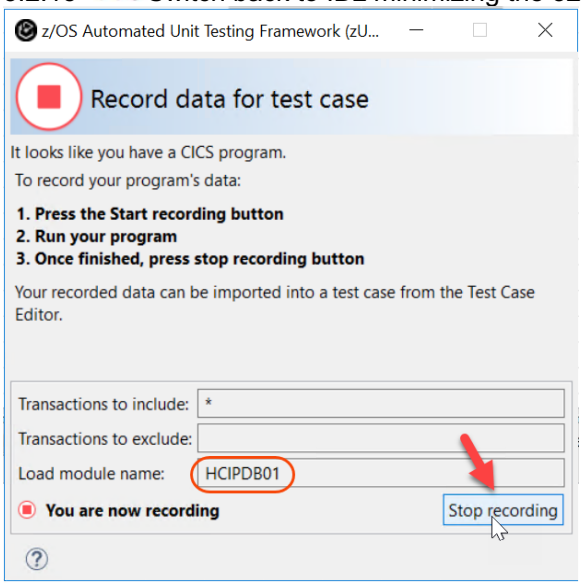
3.2.13 ⏭ Switch to the 3270 terminal emulator (that is minimized), type **2** (Inquire Patient) and press **enter**.



3.2.14 Type **1** and press **enter**

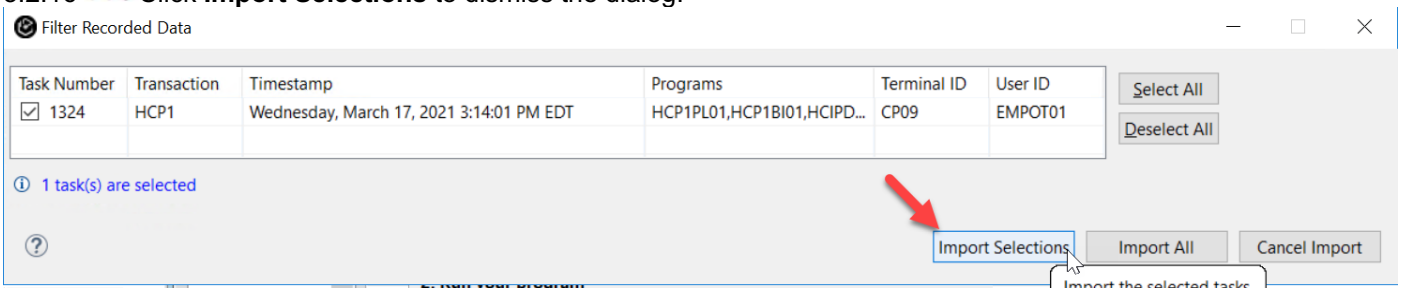


3.2.15 Switch back to IDz minimizing the 3270 screen and click **Stop recording**.



The dialog Filter Recorded Data is displayed.

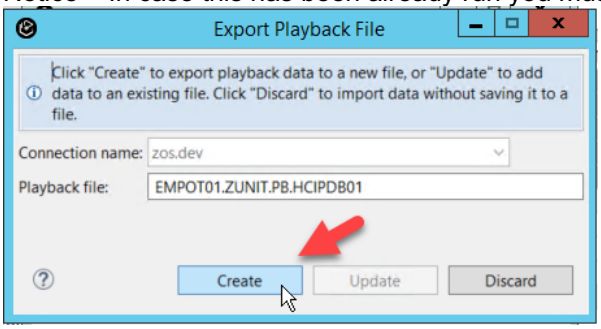
3.2.16 Click **Import Selections** to dismiss the dialog.



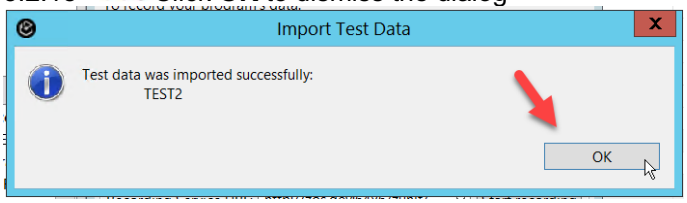
3.2.17 Change the *Playback file* to **`EMPOT01.ZUNIT.PB.HCIPDB01`** (instead of **`EMPOT05`**)

Click **Create** to export playback data to a z/OS data set

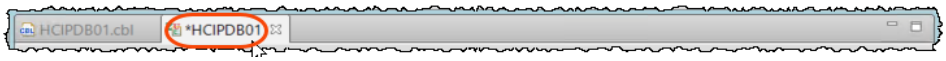
Notice – In case this has been already run you must select **Update** instead.



3.2.18 Click **OK** to dismiss the dialog

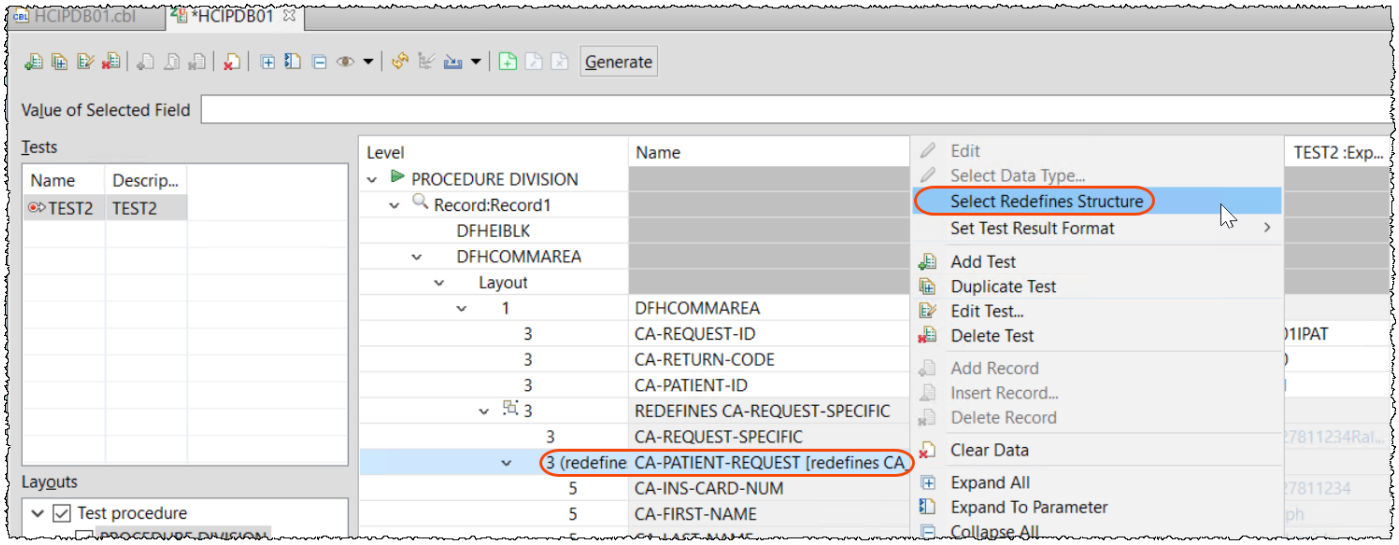


3.2.19 Double click on the **HCIPDB01** title to enlarge the view.

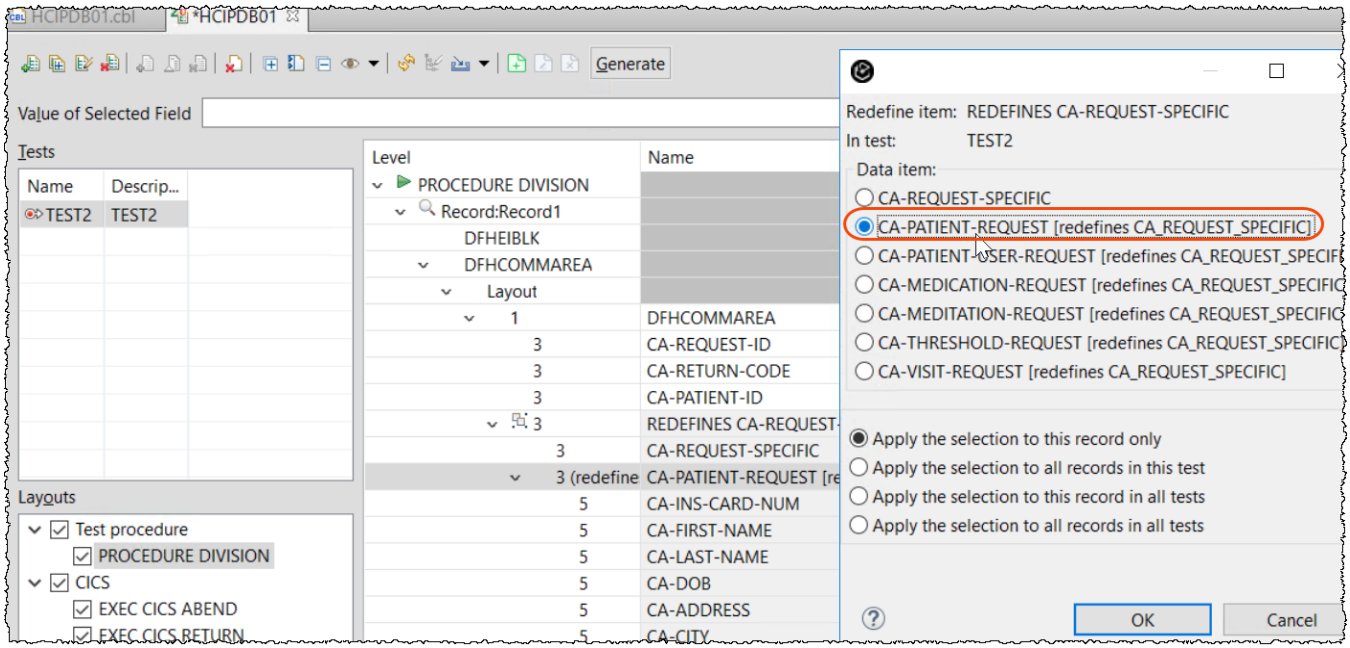


3.2.20 You now see a new test created in the editor and populated with the values from the live run.
Also you may notice that this program has many redefines.
You need to identify which is the area being used on this program execution. In our example is the “CA-PATIENT-REQUEST” area being redefined.

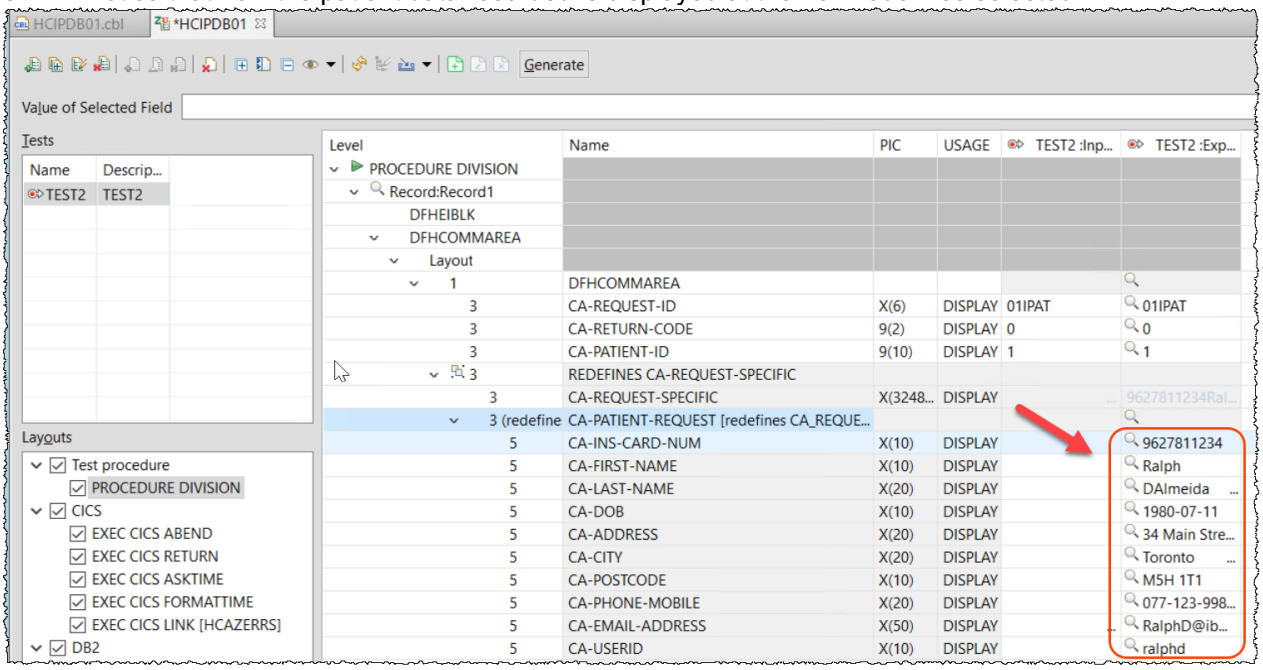
Find **“(redefines) CA-PATIENT-REQUEST**” (the first REDEFINES).
Right click on it and click **Select Redefines Structure**



3.2.21 Select **CA-PATIENT-REQUEST** and click **OK**



3.2.22 Notice that now the patient data recorded is displayed at the new redefines selected.



3.2.23 Press **Ctrl+S** to save any changes. Notice this save takes a while since values are saved on z/OS

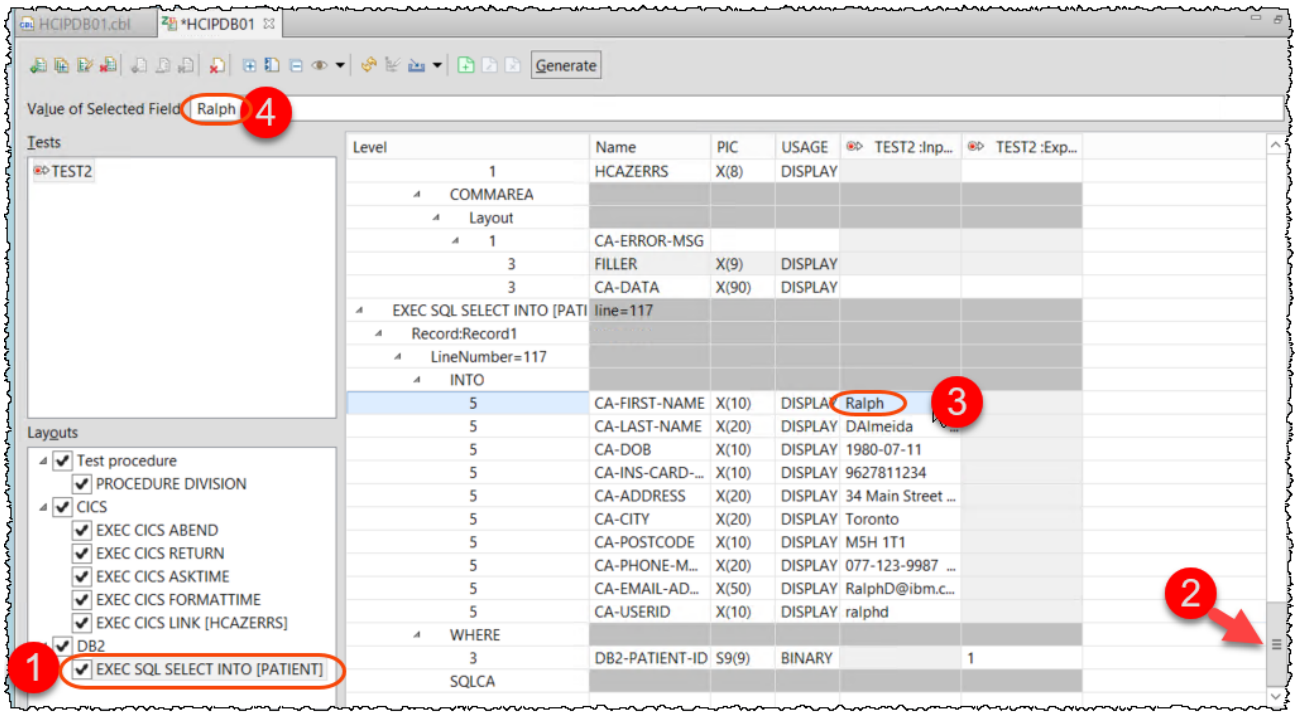
3.2.24 You now see a new test created in the editor and populated with the values from the live run.

Scroll down the editor and notice the data displayed on the screen that was captured..

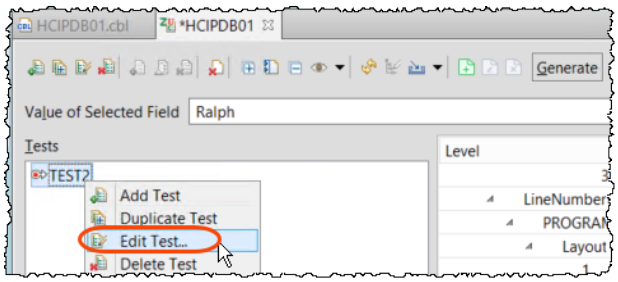
3.2.25 1 Select **EXEC SQL SELECT INTO (PATIENT)**to position the data layout for this statement.

2 Use the scroll bar to scroll down until the end,

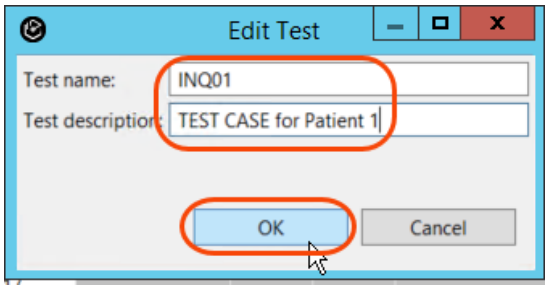
3 Click on **Ralph** . and notice that value is displayed on the line 4 "



3.2.26 Right click on **TEST2** and select **Edit Test**

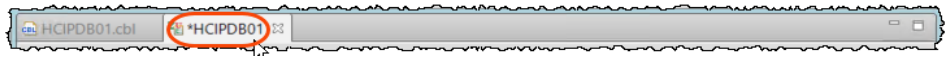


3.2.27 It is a good practice give a name for the test case.
Use a name like **INQ01** and a description like "**TEST CASE for Patient 1**": . Click **OK**



3.2.28 Press **Ctrl+S** to save any changes.

3.2.29 **Double click** again on the ***HCIPDB01*** title to shrink the view.

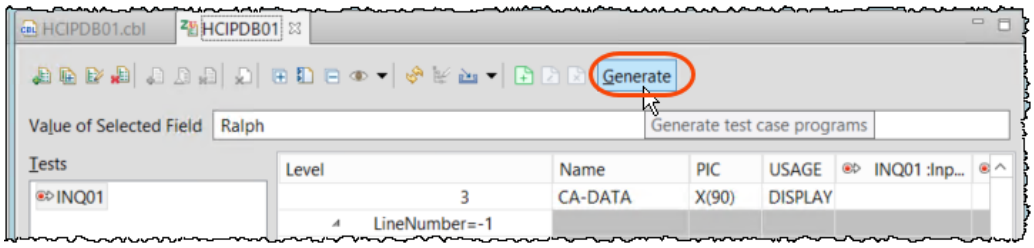


Section 4. Generate, build and run the unit test.

You will generate, build, and run the unit test.

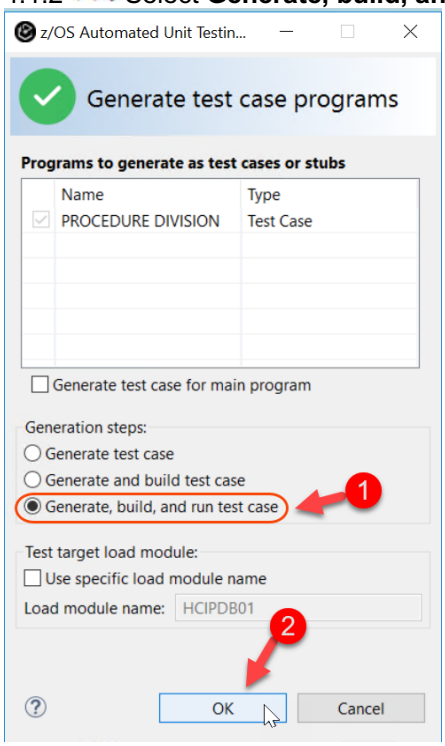
___ 4.1 Generating, building and running the test case program.

4.1.1 Now that the expected input and output values have been set in the test case editor from the recorded run, **click** on **Generate** to generate the test case program.

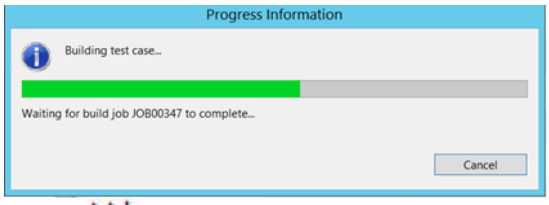


On the **Generate test case programs** dialog,

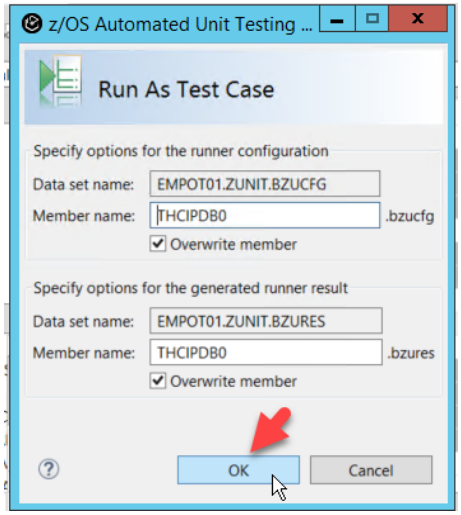
4.1.2 Select **Generate, build, and run test case** and then click **OK**.



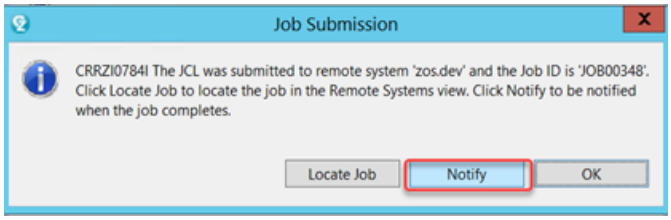
A **Progress information** dialog on the building will open.
This generation will take a while and a job is submitted to z/OS for execution. The COBOL test cases programs were compiled/link edited and are ready to run.



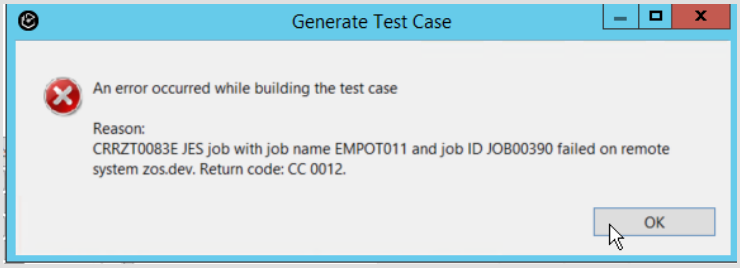
4.1.3 Once the building is complete, a **Run As Test Case** dialog will open, click **OK** to continue.



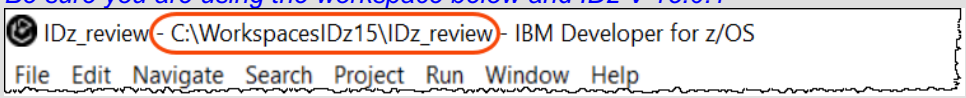
4.1.4 A **Job Submission** dialog opens, click **Notify** to be notified of job completion.



#1. Had a return code 12 as below?



You might be using a wrong IDz Workspace…
Be sure you are using the workspace below and IDz V 15.0.1

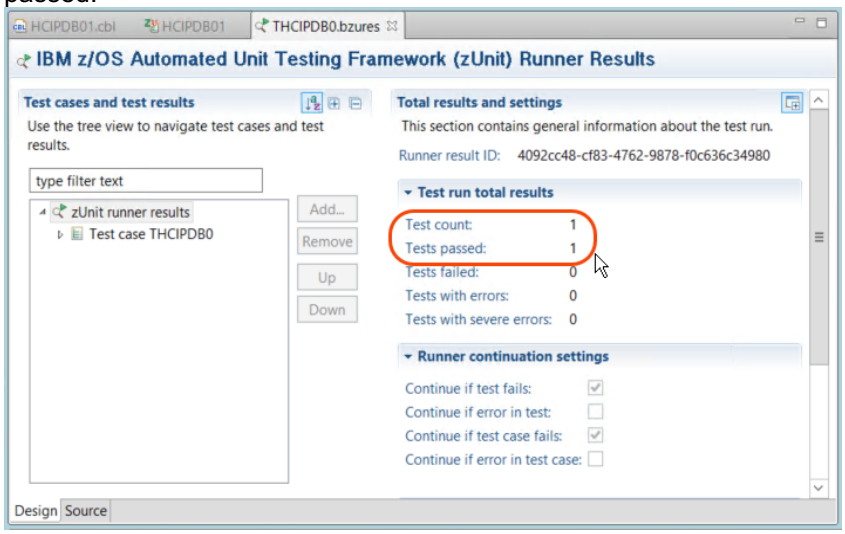


Call the instructors and
1. Close all opened editors.
2. Delete the data set EMPOT01.ZUNIT.PB.HCIPDB01 *and go back to record and try again*

#2. Had a "strange Test fail" ?

Call the instructor..
1. Delete the data set EMPOT01.ZUNIT.PB.HCIPDB01
2.All members from EMPOT01.ZUNIT.* *need to be deleted and the record started again.*

4.1.5 Once the unit test run has completed, the results screen is displayed showing test cases ran and passed.



You have now created a test case with data imported from a recorded run and have been successful in testing a CICS program without the need of a CICS environment.

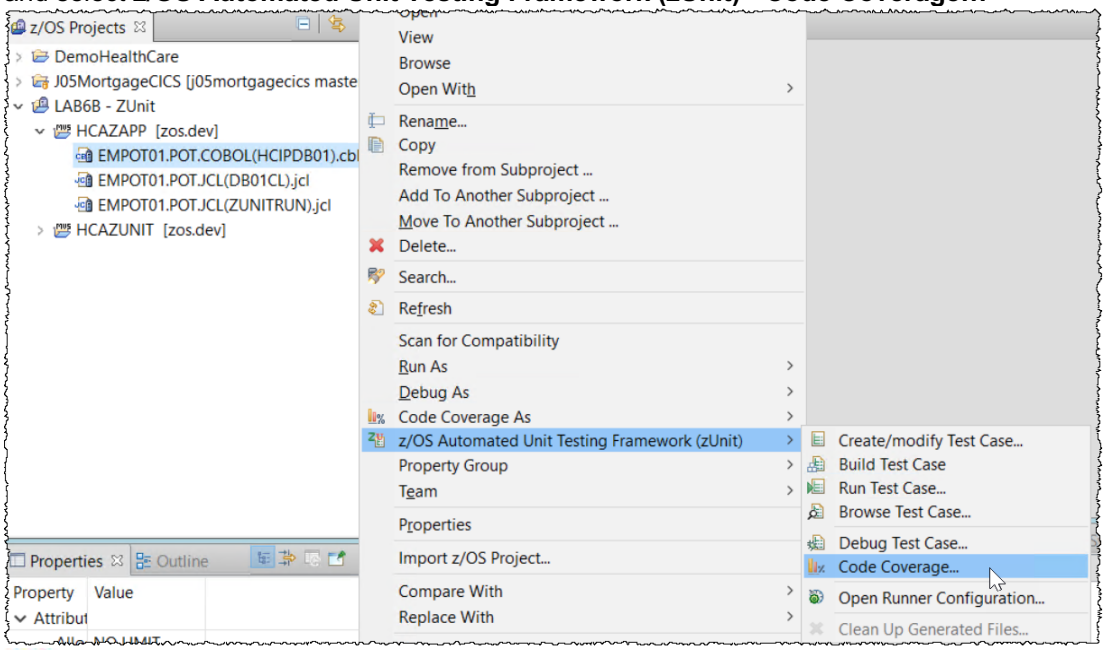
4.1.6 Use **Ctrl + Shift + F4** to close all opened editors.

___ 4.2 Running zUnit test case with Code Coverage

Code coverage analyzes a running program and generates a report of lines that are executed, compared to the total number of executable lines. Sometimes you need to be sure that your test is covering all capabilities of your program and that you are having the correct test cases.

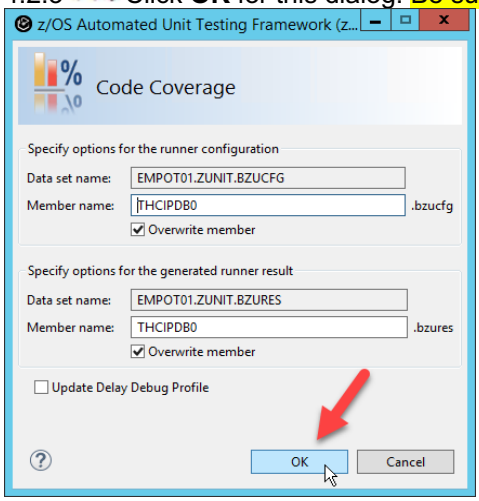
Notice that you are doing the code coverage using a batch job without need to have CICS and DB2 active, what is very handy..

4.2.1 Right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl** and select **z/OS Automated Unit Testing Framework (zUnit)->Code Coverage…**



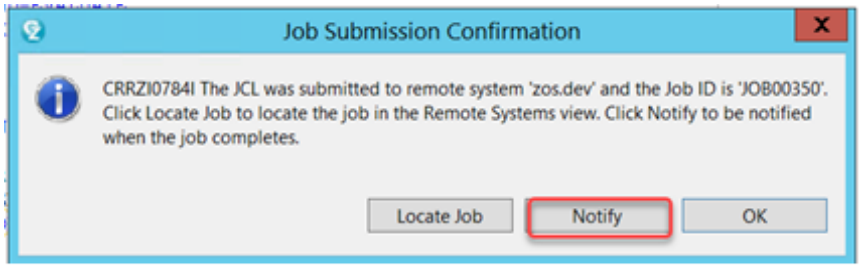
If you get a message that files are updated click **Yes** to continue

4.2.3 Click **OK** for this dialog. Be sure that *Update Delay Debug Profile* is **NOT** checked.

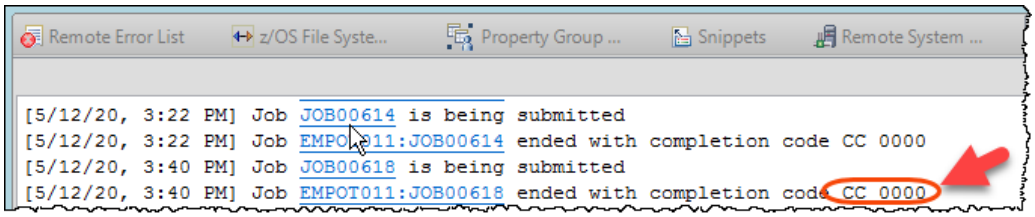


4.2.4 A Job Is submitted for batch execution..

Click **Notify** on the Job Submission Confirmation dialog.

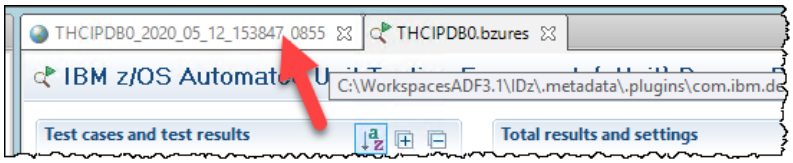


Make sure the job completes with completion code of **0**.



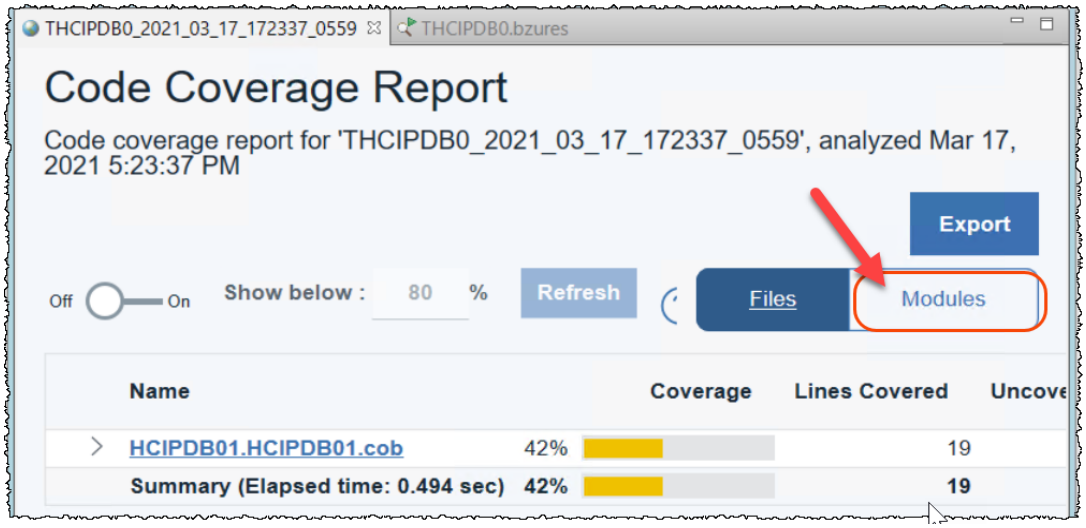
4.2.5 Once again the zUnit Results are displayed and the Test passed.

Click on **THCIPDB0_yyyy_mm_dd_xxxxxxx** tab to see the *Code Coverage* report

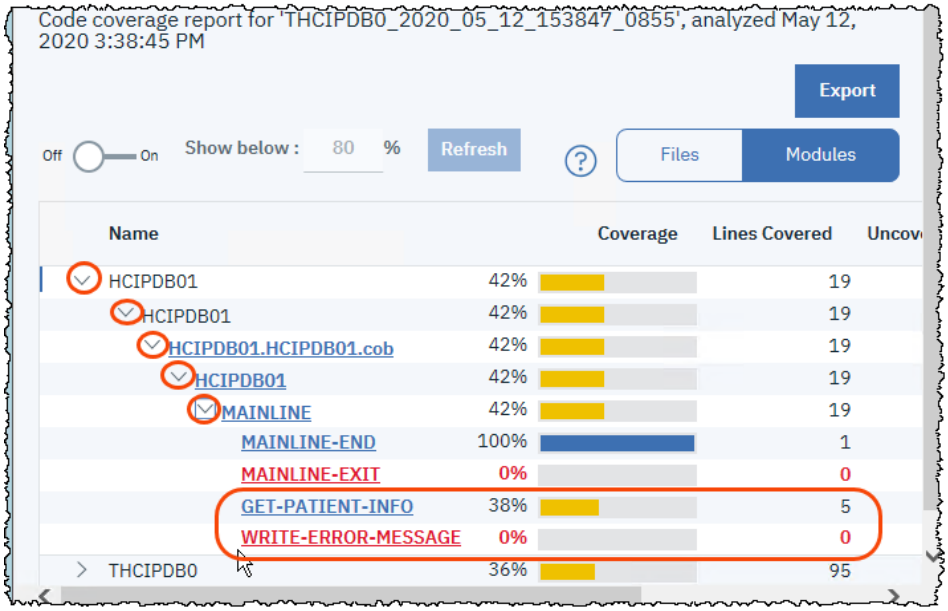


4.2.6 The code coverage report shows all COBOL programs executed for this specific test case.
It shows the coverage of the COBOL programs generated for zUnit to perform the test as well our *HCIPDB01* COBOL program..

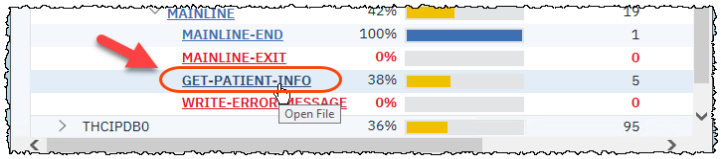
Since we are just interested in the code coverage of program being tested click on **Modules**:



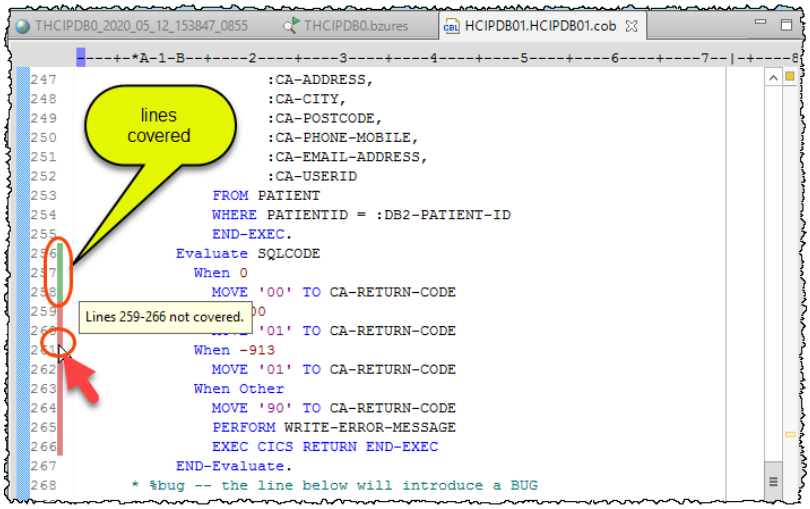
4.2.7 **Expand the nodes** as below to see the COBOL paragraphs.
Notice that *GET-PATIENT-INFO* has 38% of coverage and that *WRITE-ERROR-MESSAGE* was not covered at all on this test case.



4.2.8 Click on **GET-PATIENT-INFO** to see more details



4.2.9 **Scroll down** and **move the mouse to the colored areas on left**.
The lines covered by this test case are in **green** and the lines not covered are in **red.**
Also from the previous image we can see the WRITE-ERROR_MESSAGE paragraph is not covered at all.



4.2.10 From this report we can see that the test cases created for this program are not sufficient ..
The developer could go back to the test cases editor (step 3.2.19 above) and manually add test cases that cover other paragraphs. Due the lack of time we will not cover that here, but if you have extra time you can try it. Ask the instructor for guidance.

Section 5. Introduce a bug in the program and rerun the unit test.

Using IDz you will modify the program and introduce a bug. Then you will rerun the unit test created in the previous section.

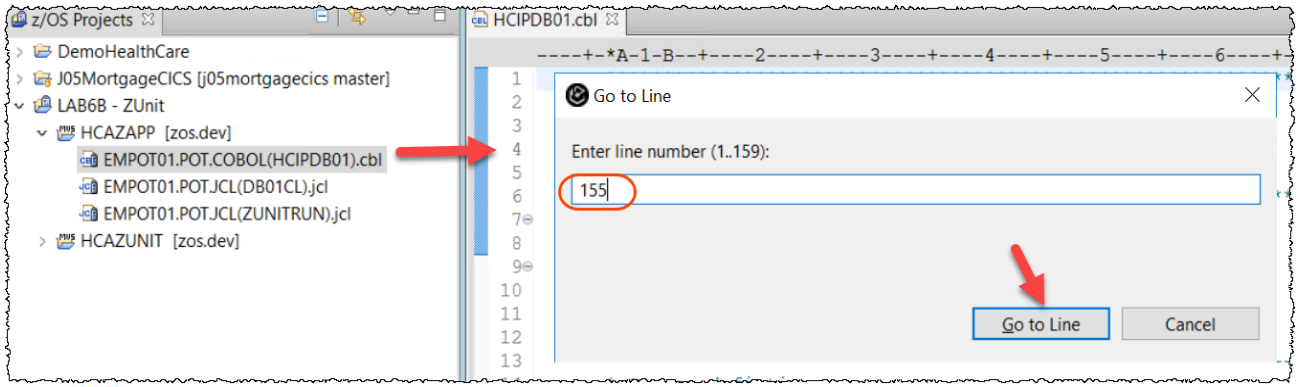
___ 5.1 Modifying the program and introduce a bug

.

5.1.1 ⏭ On IDz, close all active windows by pressing **Ctrl+Shift+F4**

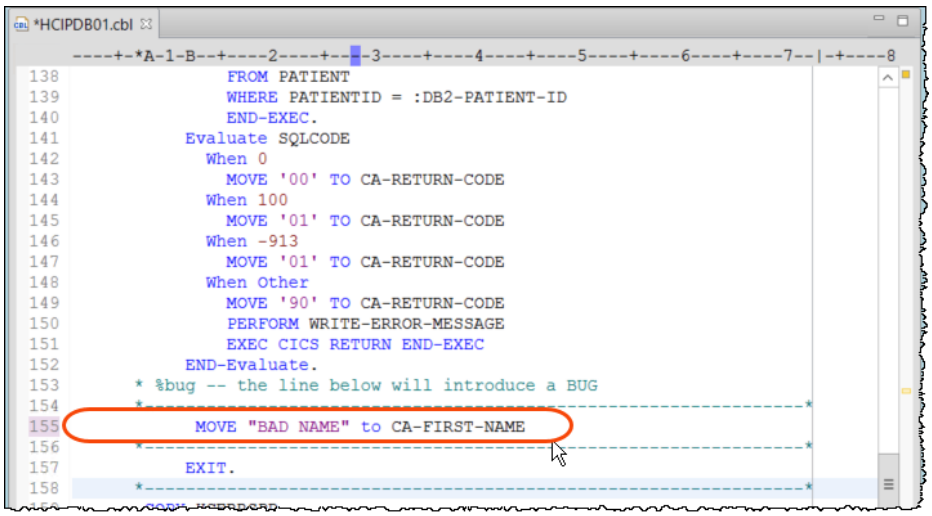
⏭ Open **EMPOT01.POT.COBOL (HCIPDB01).cbl** by double clicking on it in the z/OS Projects view.

5.1.2 ⏭ On the editor, go to line 155 by pressing **Ctrl+L** and entering **155** and clicking **Go to Line**.



5.1.3 ⏭ Change the line 155 removing the * from the statement that moves “BAD NAME”,

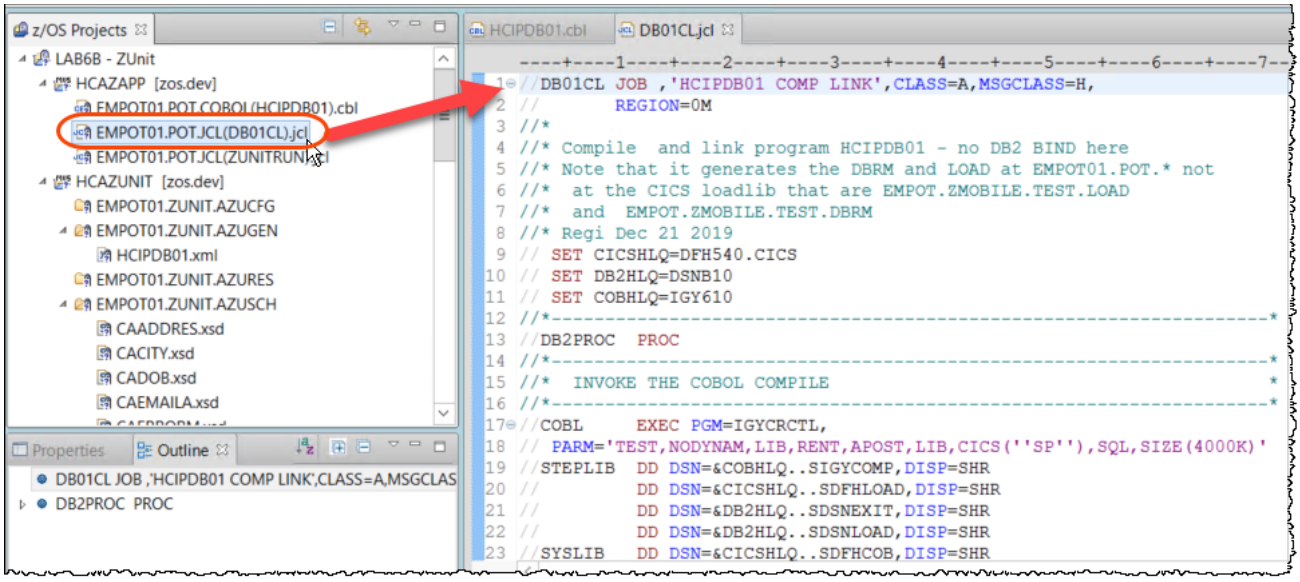
⏭ Press **Ctrl+S** to save the changes.



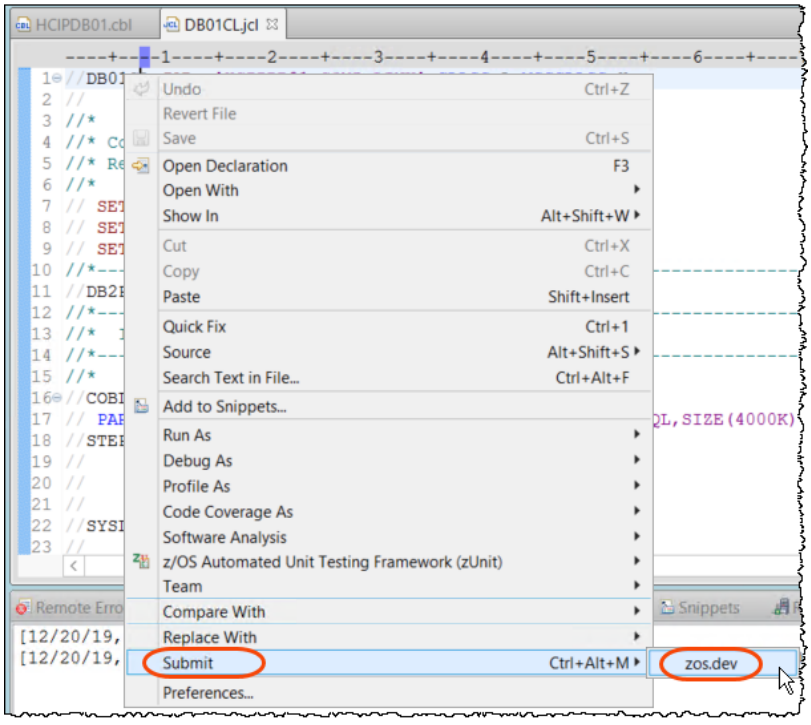
___ 5.2 Rebuilding the changed program without deploying to CICS

.5.2.1 On the z/OS Projects view, double click **EMPOT01.POT.JCL (DB01CL).jcl,** to edit .

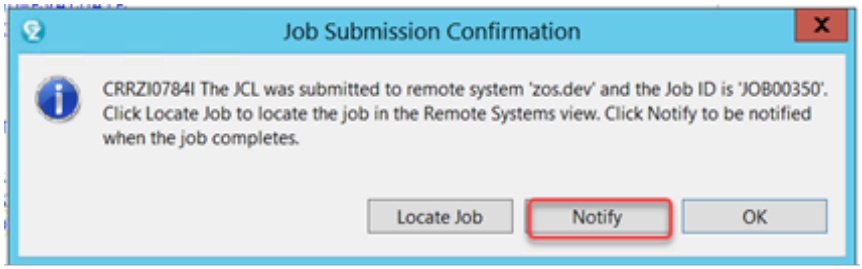
This JCL will compile and link the changed COBOL program *HCIPDB01* using a working PDS to store the load module.
Notice that DB2 bind is not necessary since DB2 SQL calls will be intercepted when running the generated zUnit test cases.



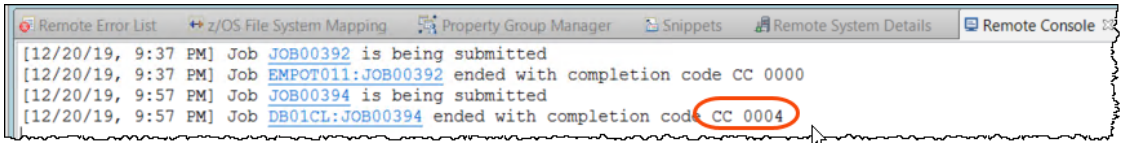
5.2.2 Right click and select **submit > zos.dev** to submit this job for execution



5.2.3 Click **Notify** on the Job Submission Confirmation dialog.

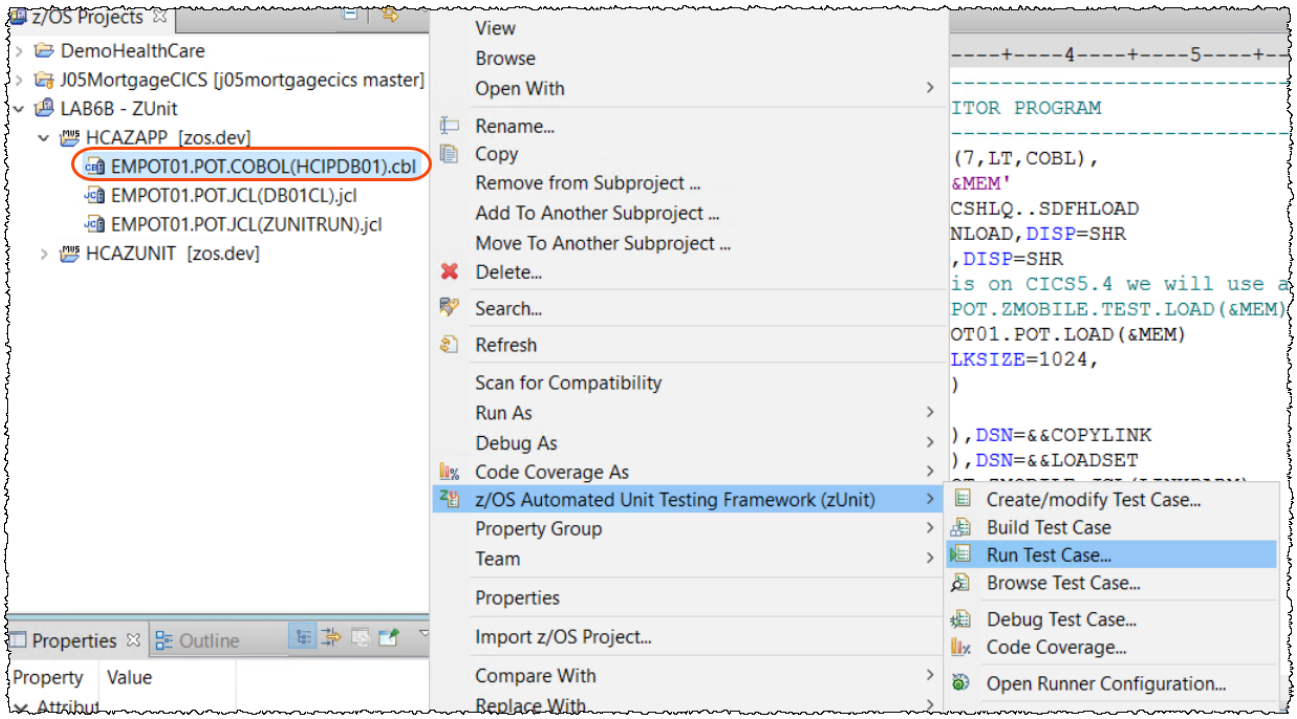


Make sure the job completes with completion code of **4**.

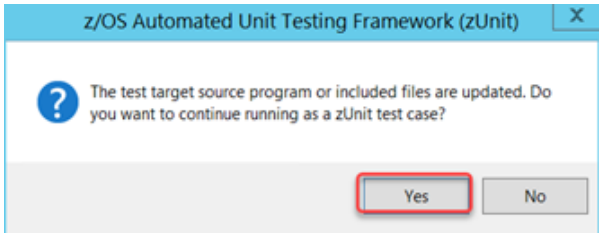


___ 5.3 Rerunning the test case

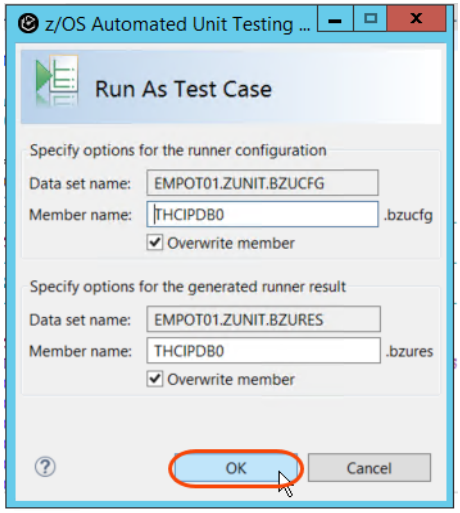
5.3.1 On the z/OS Projects view, select **EMPOT01.POT.COBOL(HCIPDB01).cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)->Run Test Case..** action.



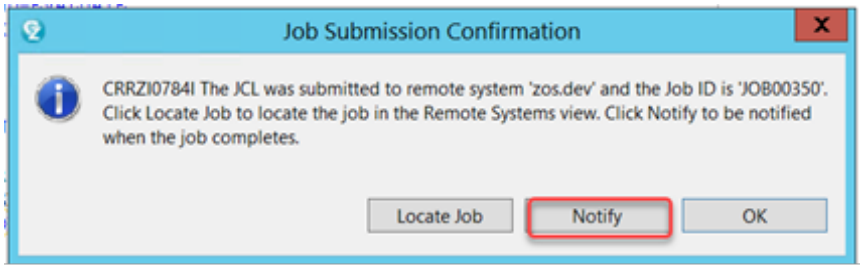
5.3.2 Click **Yes** to the following dialog to continue running the test case.



5.3.3 Click **OK** to the **Run As Test Case** dialog.

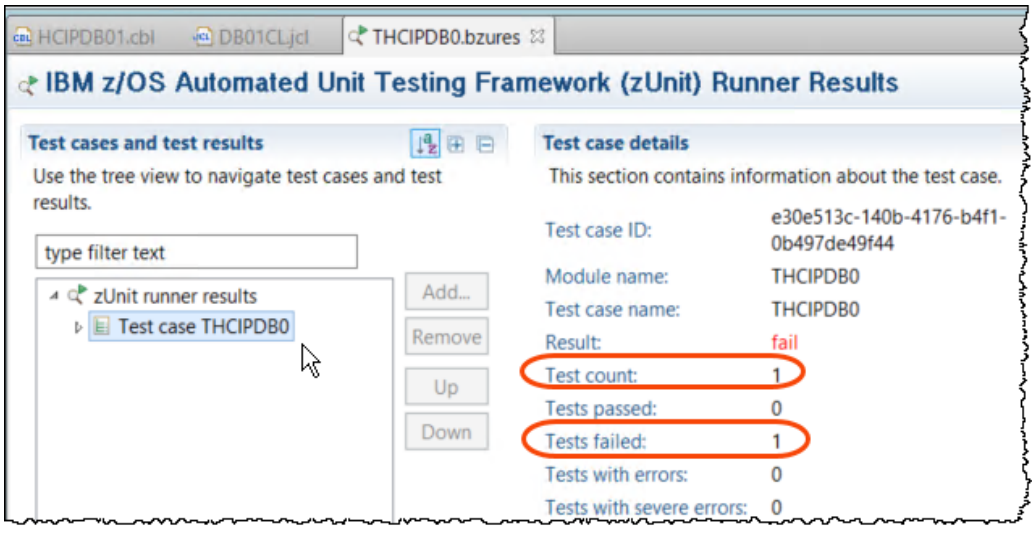


5.3.4 Click **Notify** on the Job Submission Confirmation dialog.

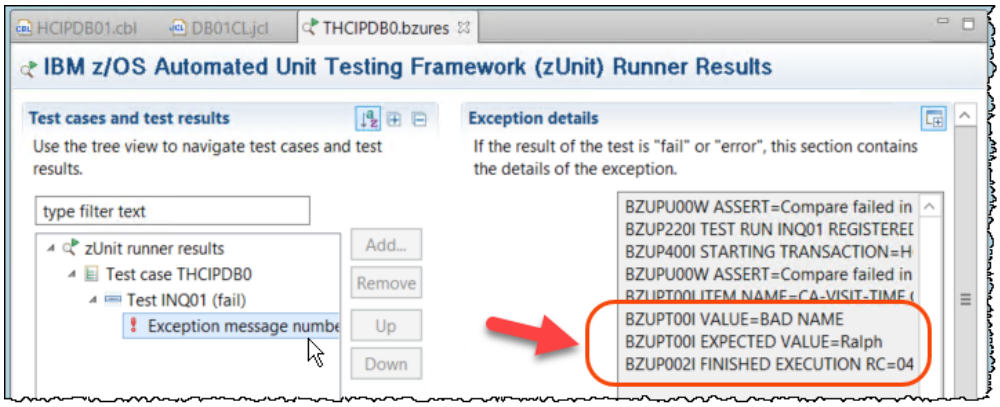


5.3.5 When the test run completes, the test results will be displayed, and it showed 1 test was ran and it failed.

Click **Test case THCIPDB0.** The failure is expected because the expected value of the error message is different from the actual value.



5.3.6 Expand **Test case THCIPDB0, Test INQ01 (fail)** and click **Exception message number** to verify the value received versus the expected value and verify the failure



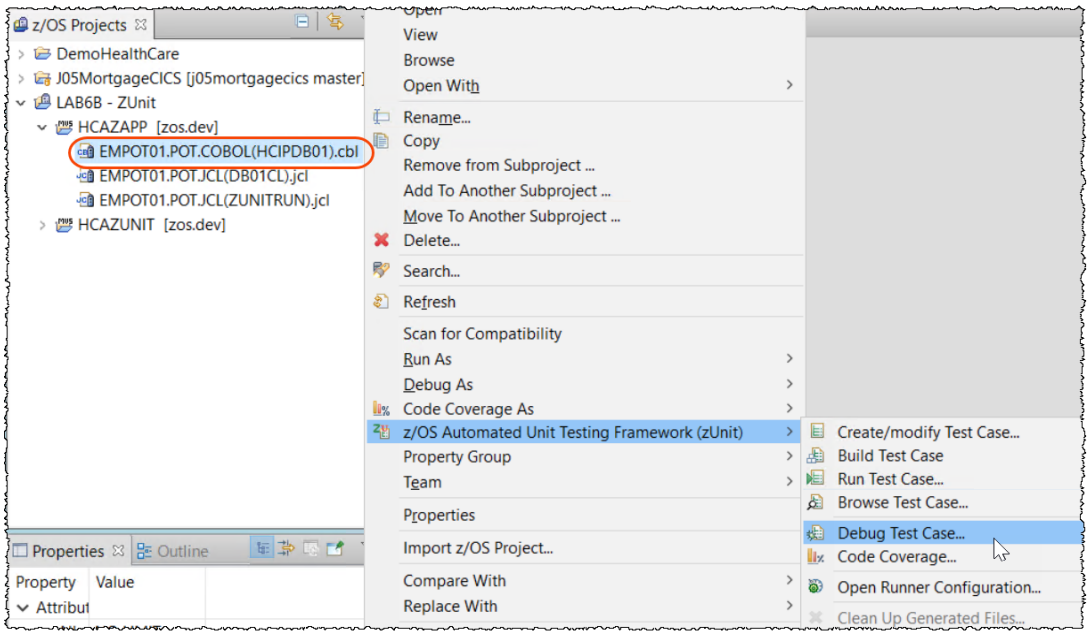
___ 5.4 Running zUnit test case with debugging

The Debug dialog will debug the test case. Note that will debug the zUnit programs, the generated COBOL and also the COBOL program that you are testing. Below one example.

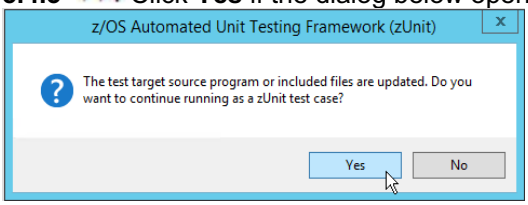
Notice that you are debugging a batch job without need to have CICS and DB2 active, what is very handy.

5.4.1 Close all active windows by pressing **Ctrl+Shift+F4**

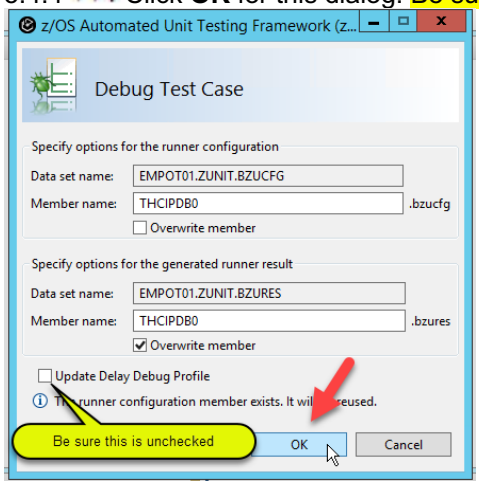
5.4.2 Right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl**
and select **z/OS Automated Unit Testing Framework (zUnit) > Debug Test Case…**



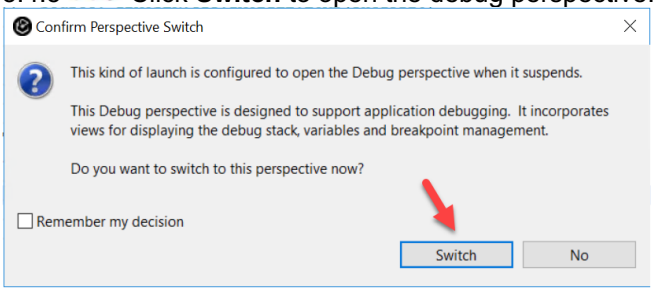
5.4.3 Click **Yes** if the dialog below opens



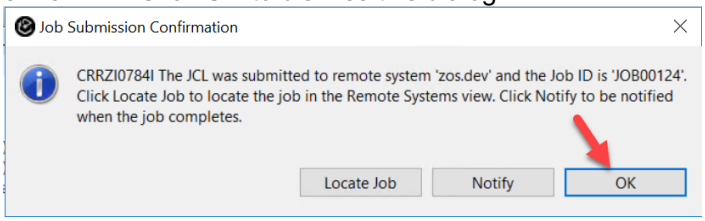
5.4.4 Click **OK** for this dialog. Be sure that Update Delay Debug Profile is un-checked.



5.4.5 Click **Switch** to open the debug perspective.

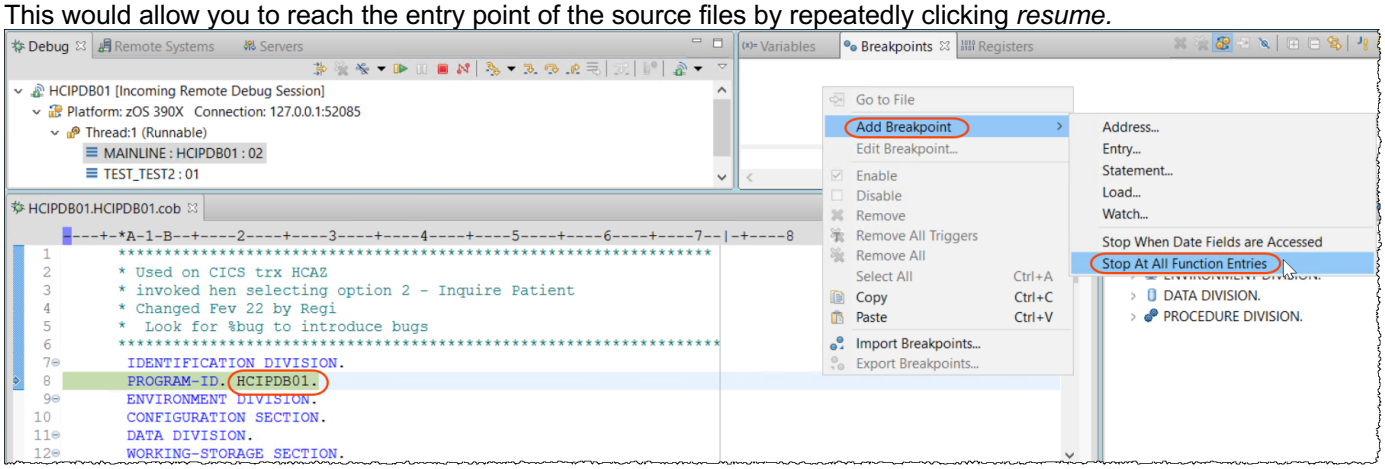


5.4.6 Click **OK** to dismiss this dialog

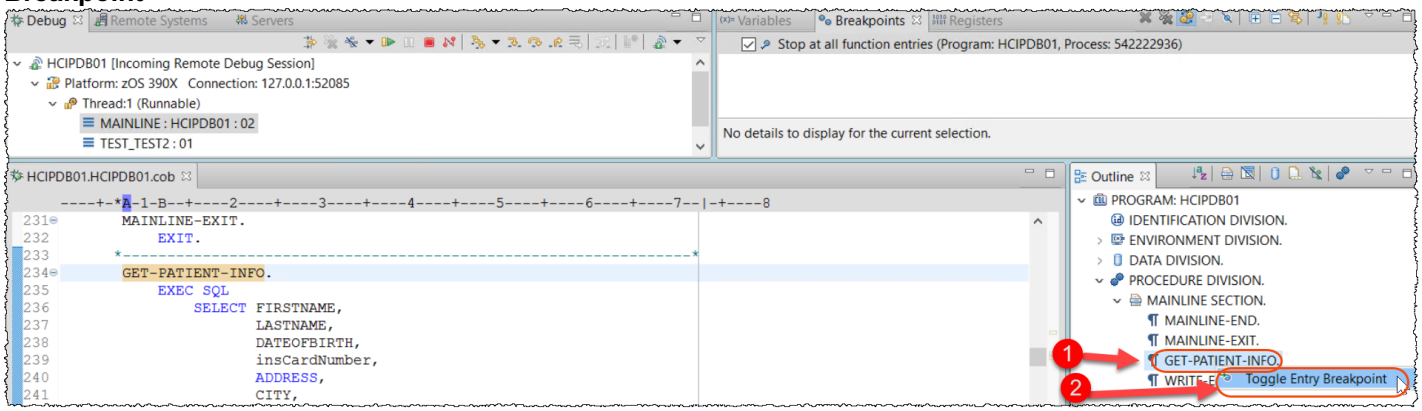


5.4.7 Notice that the first program being debugged is the program that you introduced the bug. (**HCIPDB01**)
The execution is stopped BEFORE that program starts. You can add some breakpoints before running.

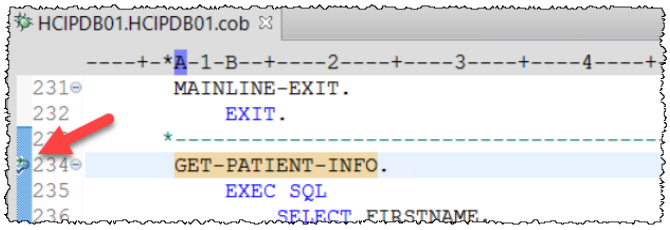
Using the Breakpoints view, right click and select **Add Breakpoints > Stop At All Functions Entries**.
This would allow you to reach the entry point of the source files by repeatedly clicking *resume.*



5.4.8 Using the Outline view to navigate to **GET-PATIENT-INFO** and right click and select **Toggle Entry Breakpoint**

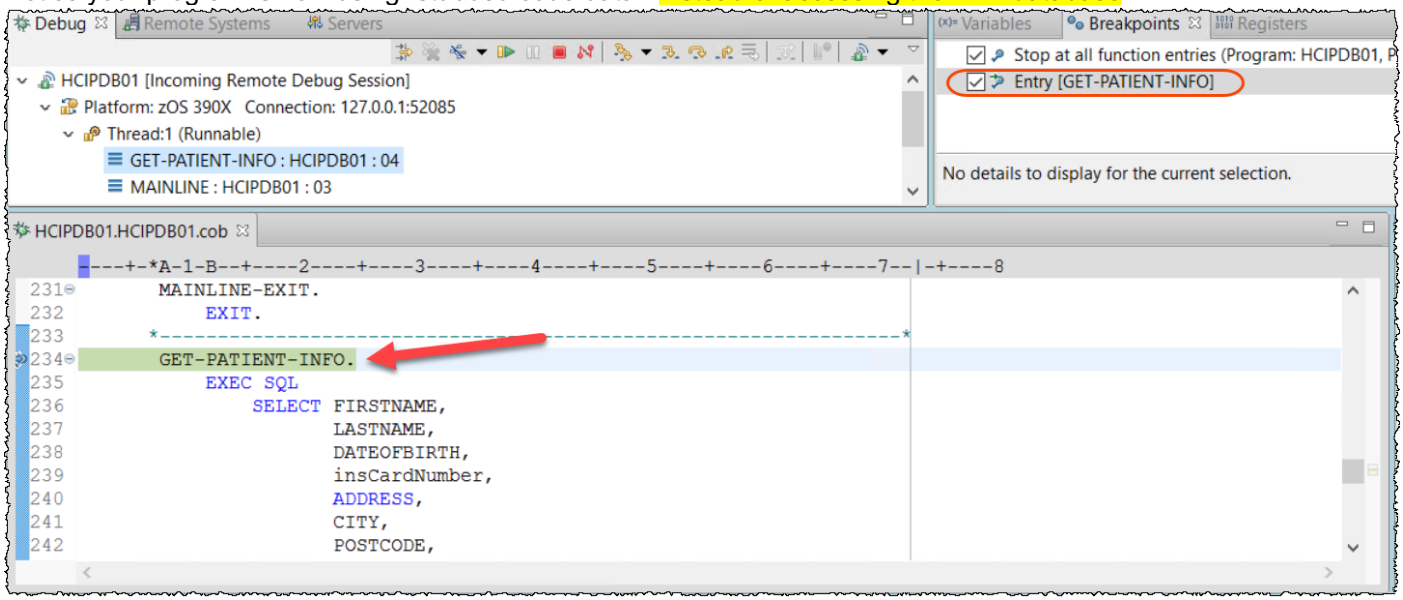


5.4.9 A breakpoint is created on the *EXEC SQL SELECT* statement. When the program runs it will stop there.



5.4.10 **Click** on icon or press **F8** to *resume*. the execution

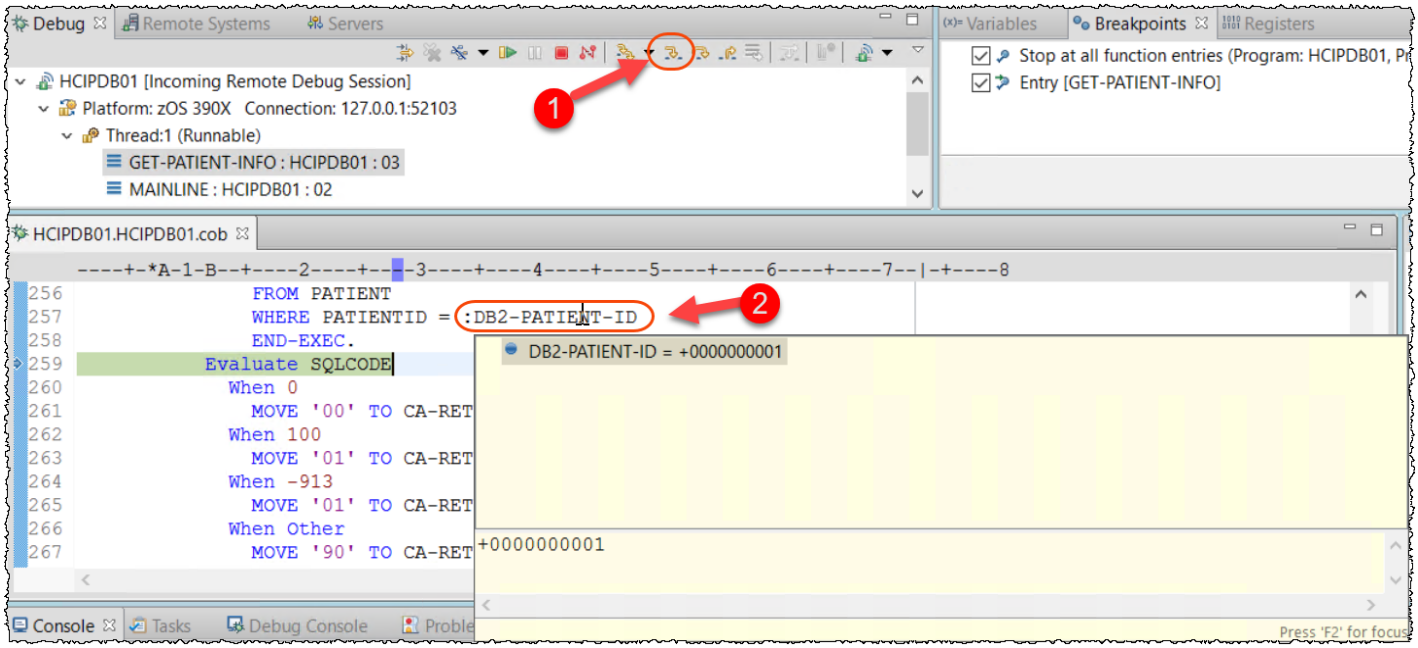
5.4.11 The execution will halt at the ***SQL SELECT*** statement due the breakpoint you added.
Notice your program is now using “stubbed code data” instead of accessing the DB2 database..



5.4.12 **Click** on icon or press **F5** to *Step Into* the code until you find the statement below.

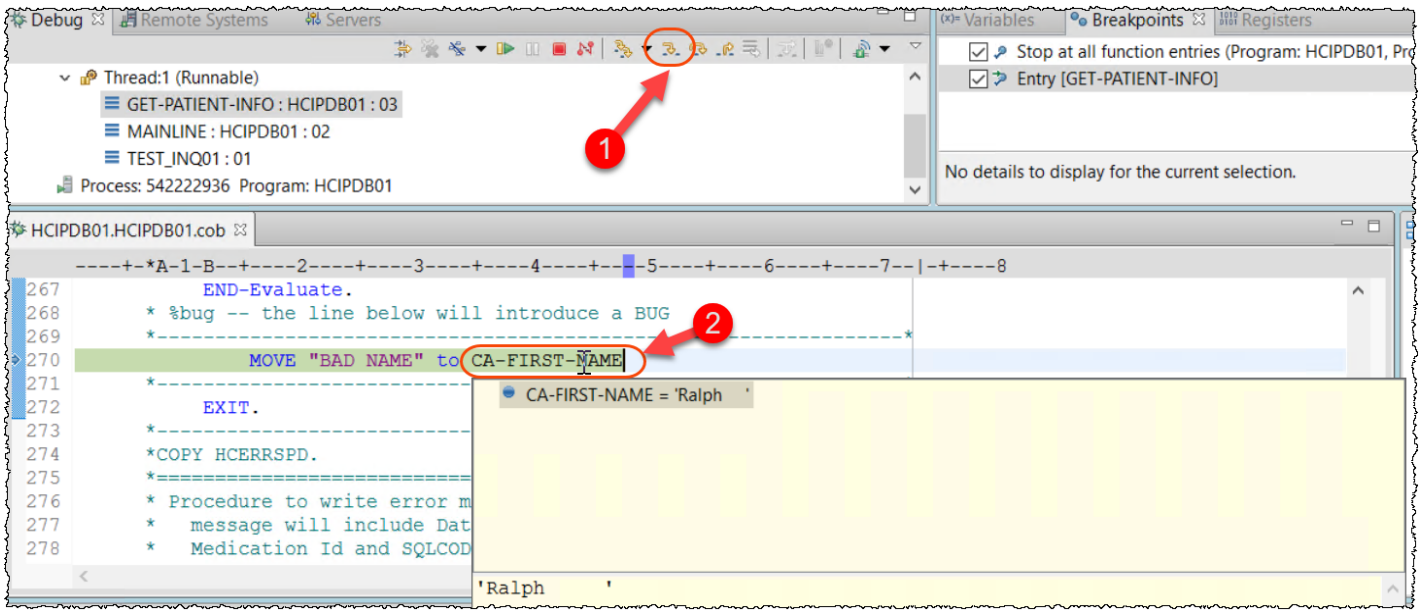
Move the mouse to **: DB2-PATIENT-ID** and see it contents

Again, notice that we get DB2 data without going to the database, but using the stub recorded earlier (Play Back file).

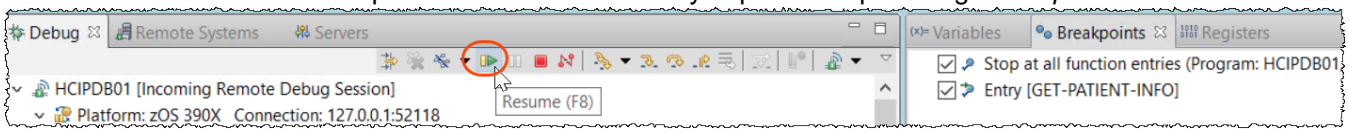


5.4.13 Keep clicking on icon or press **F5** to *Step Into* the code until you see the bug that you had introduced.

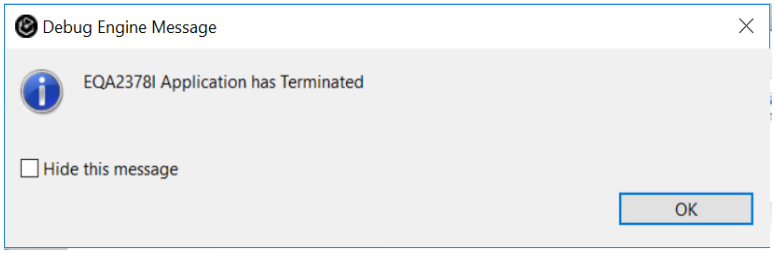
You also can see the BUG that you introduced. Move the mouse to **CA-FIRST-NAME** to see the field content (**`Ralph`**), which will be replaced by **`"BAD NAME".`**



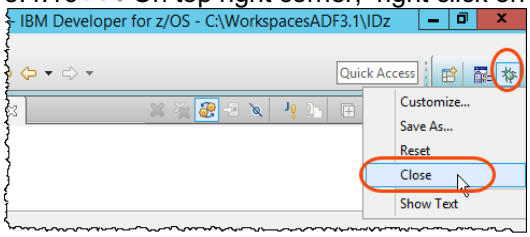
5.4.14 Click on icon or press **F8** to *resume*.. Or if you prefer keep clicking on *Step Into*.



5.4.15 You will see that the debug ends when you have the dialog below. Click **OK**



5.4.16 On top right corner, right click on icon and select **Close** to close the debug perspective



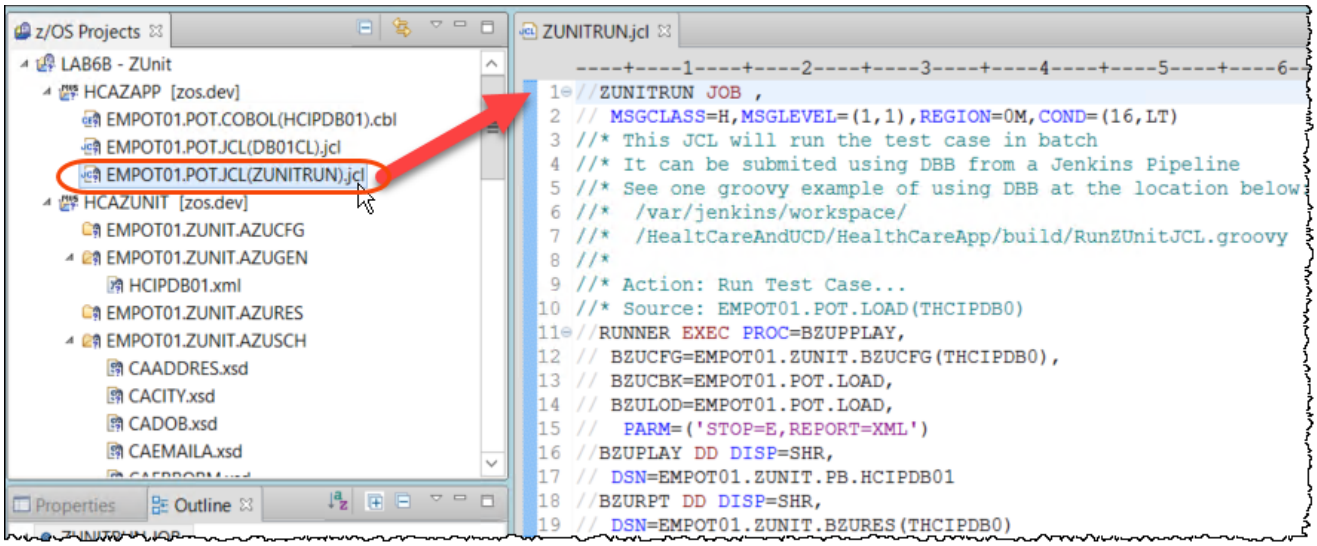
Section 6. Run the unit test from a batch JCL.

Once a test case has been generated and built, it can be executed using a batch run via JCL.
This would enable it to be ran as part of an automated process or pipeline..

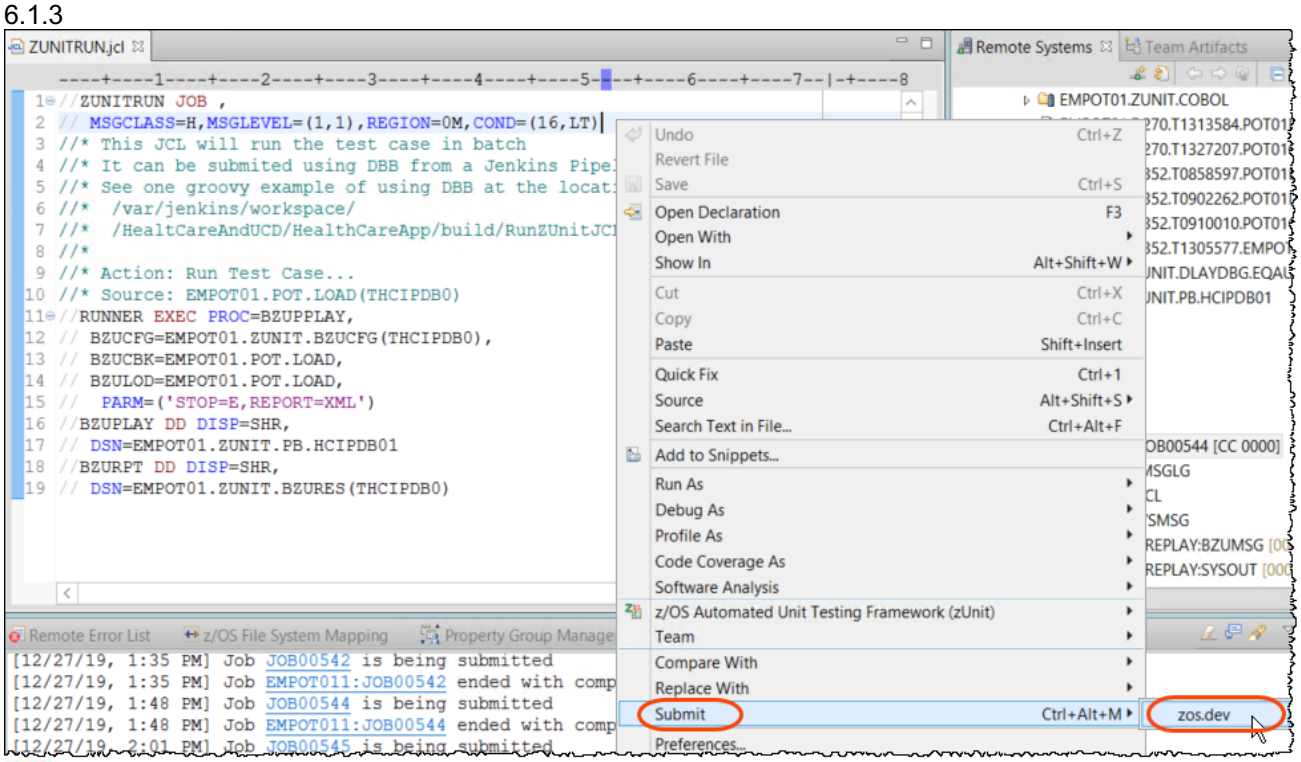
___ 6.1 Running the unit test from a batch JCL

6.1.1 Close any active windows by pressing **Ctrl+Shift+F4**.

6.1.2 On the IDz **z/OS Projects** view, double click **EMPOT01.POT.JCL(ZUNITRUN).jcl** to open it.
This JCL will run the test case that you created using a batch job.

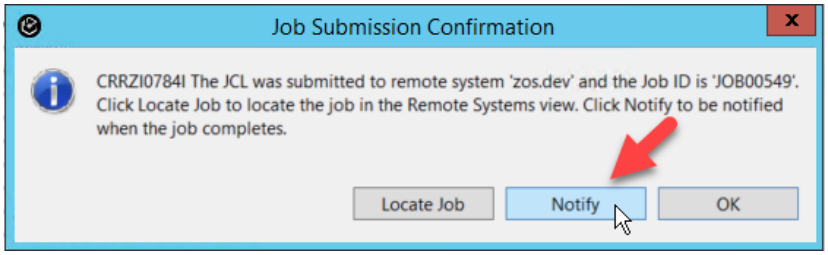

z/OS Projects
LAB6B - ZUnit
HCAZAPP [zos.dev]
EMPOT01.POT.COBOL(HCIPDB01).cbl
EMPOT01.POT.JCL(DB01CL).jcl
EMPOT01.POT.JCL(ZUNITRUN).jcl
HCAZUNIT [zos.dev]
EMPOT01.ZUNIT.AZUCFG
EMPOT01.ZUNIT.AZUGEN
HCIPDB01.xml
EMPOT01.ZUNIT.AZURES
EMPOT01.ZUNIT.AZUSCH
CAADDRES.xsd
CACITY.xsd
CADOB.xsd
CAEMAILA.xsd
Properties
Outline
ZUNITRUN.jcl
1 //ZUNITRUN JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),REGION=0M,COND=(16,LT)
3 //* This JCL will run the test case in batch
4 //* It can be submited using DBB from a Jenkins Pipeline
5 //* See one groovy example of using DBB at the location below
6 //* /var/jenkins/workspace/
7 //* /HealtCareAndUCD/HealthCareApp/build/RunZUnitJCL.groovy
8 //*
9 //* Action: Run Test Case...
10 //* Source: EMPOT01.POT.LOAD(THCIPDB0)
11 //RUNNER EXEC PROC=BZUPPLAY,
12 // BZUCFG=EMPOT01.ZUNIT.BZUCFG(THCIPDB0),
13 // BZUCBK=EMPOT01.POT.LOAD,
14 // BZULOD=EMPOT01.POT.LOAD,
15 // PARM=('STOP=E,REPORT=XML')
16 //BZUPLAY DD DISP=SHR,
17 // DSN=EMPOT01.ZUNIT.PB.HCIPDB01
18 //BZURPT DD DISP=SHR,
19 // DSN=EMPOT01.ZUNIT.BZURES(THCIPDB0)

6.1.3



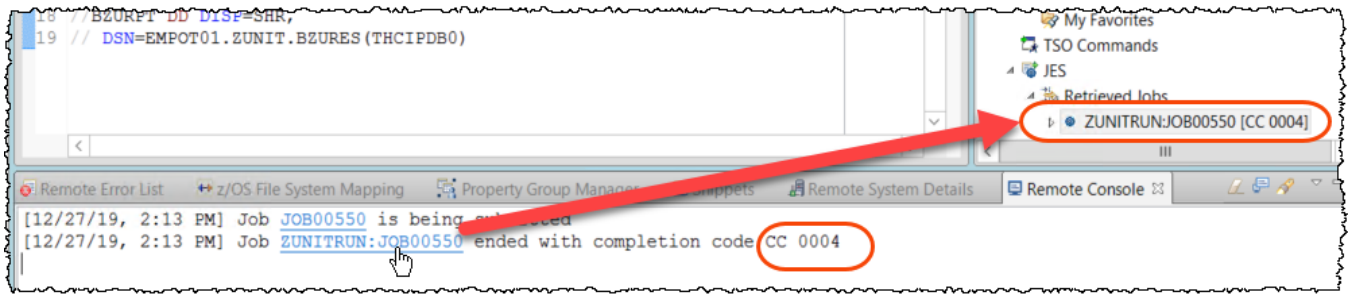
Right click the editor and select **Submit > zos.dev**.

6.1.4 Click **Notify** on the Job Submission Confirmation dialog.

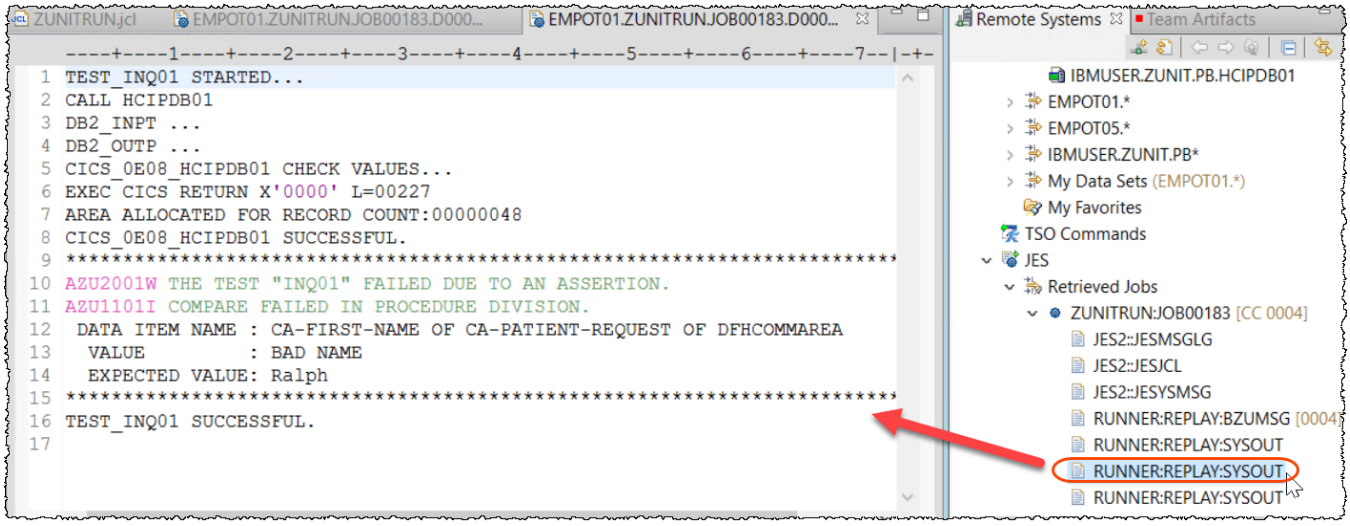


6.1.5 You MUST have **004** as return code.

Click on **ZUNITRUN:JOB00xxx**

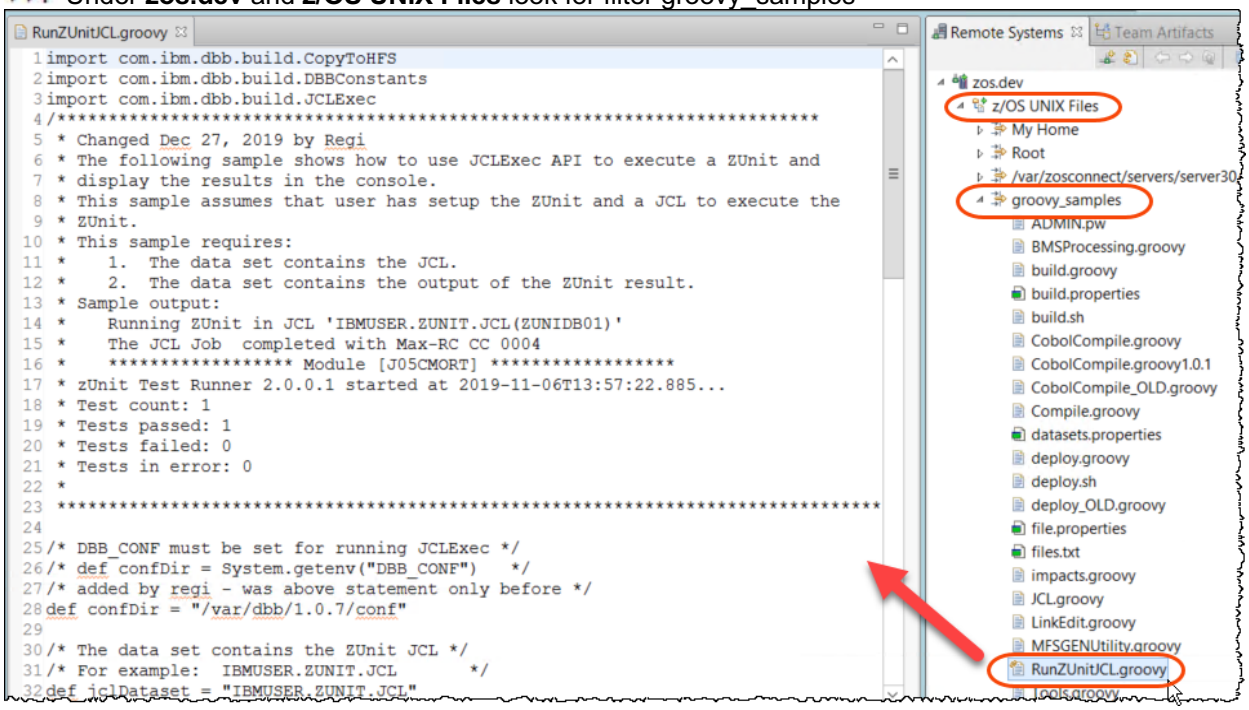


6.1.6 Expand **ZUNITRUN:JOB00xxx** and verify the results double clicking on **RUNNER:REPLAY:SYSOUT.**



6.1.7 This JCL could be executed using DBB and part of a Jenkins Pipeline.
You may see an example of a groovy script used by DBB at the USS file below:
/var/jenkins/workspace/HealtCareAndUCD/HealthCareApp/build/ RunZUnitJCL.groovy

Under **zos.dev** and **z/OS UNIX Files** look for filter groovy_samples



Notice that this capability allows to invoke zUnit using pipelines like Jenkins.

Congratulations! You have completed the Lab 2A.

LAB 2B – (OPTIONAL) Using IBM zUnit to Unit Test a COBOL/DB2 batch program (60 minutes)

Updated June 25, 2021 by Regi –(reviewed by Wilbert Kho)

This lab will take you through the steps of using the automated unit testing (zUnit) capabilities of IBM Developer for z (IDz) to create a unit test case for a COBOL/DB2 batch program. This enables the testing of just a single program using a JCL being executed. This is done by stubbing out DB2 calls, enabling the program to be tested without a DB2 environment being active. This enables a developer to test early without using DB2 and necessary bindings.

In this lab you will record interaction with a COBOL/DB2 program via JCL execution and import the recorded data into the unit test case generation process. You will build and run the unit test, make a change to the COBOL/DB2 program, and rerun the unit test.

Overview of development tasks

To complete this tutorial, you will perform the following tasks:

PART #1 – Unit test on COBOL/DB2 program DB2BATCH and introduce a bug

1. **Run the COBOL/DB2 batch program using JCL**
   → You will submit a JCL to execute a COBOL/DB2 program that calls a COBOL subprogram to print a small report.
2. **Use zUnit to record the batch execution.**
   → You will record the batch execution using the COBOL/DB2 program and subprograms.
3. **Generate, build, and run the unit test generated program**
   → You will compile and link-edit the generated unit test program, followed by running the unit test.
4. **Modify the COBOL/DB2 program (introduce a bug) and rerun the unit test**
   → You will modify the COBOL/DB2 program introduce a bug, rerun the unit test, and observe the failure of the test case.

PART #2 – Fix COBOL/DB2 program DB2BATCH and re-run the Unit test

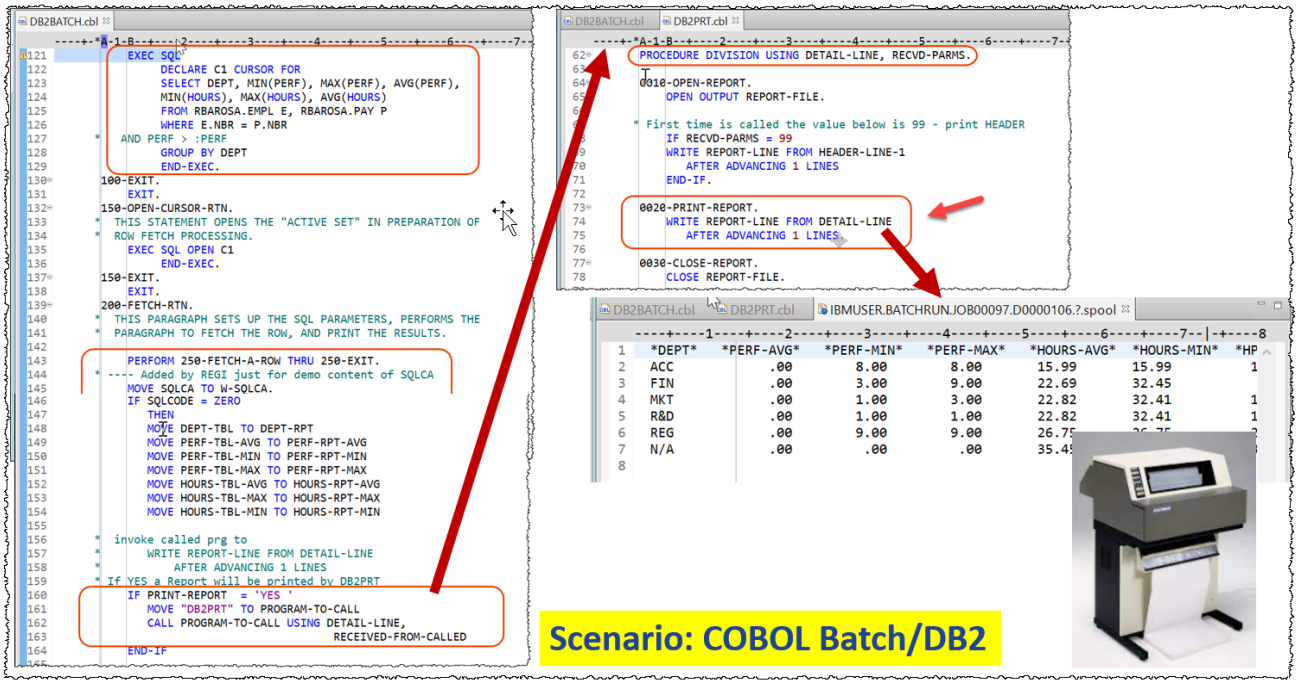
5. **Run the batch program using the provided JCL and verify the bug .**
   → You will run the Batch JCL and observe the bug on the printed report..
6. **Use IDz to fix the bug, recompile the COBOL/DB2 program**
   → The bug is fixed using IDz, program fixed is rebuilt
7. **Rerun the zUnit and verify that the bug is eliminated**
   → When the zUnit test case is executed you can verify that the program is fixed.

PART #1 – Unit test on program DB2BATCH and introduce a bug

Section 1. Run the COBOL/DB2 batch program using JCL

You will submit a provided JCL to execute on z/OS and verify the report produced by the second program that is invoked by a dynamic call.

Below it is showing the COBOL program DB2BATCH invoking the program DB2PRT that prints the report



___ 1.0 Connect to z/OS using IDz

1.0.1 Start *IBM Developer for z Systems version 15* if it is not already started otherwise go to step 1.1.1

⏭ Using the desktop double click on **IDz V15** icon.

⏭ Verify that the message indicates that it is Version **15.0.1**

IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon.

___ 1.1 Submit a provided JCL for execution

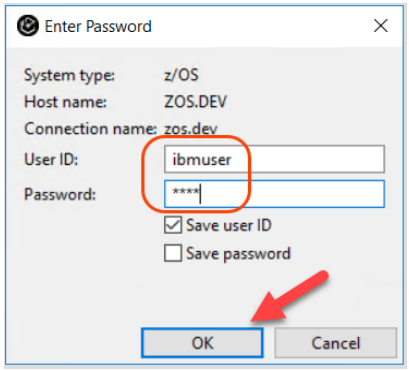
You will use IDz to submit a provided JCL .

1.1.1 ⏭ Open the **z/OS Projects** perspective by selecting
Window > Perspective > Open Perspective > z/OS Projects

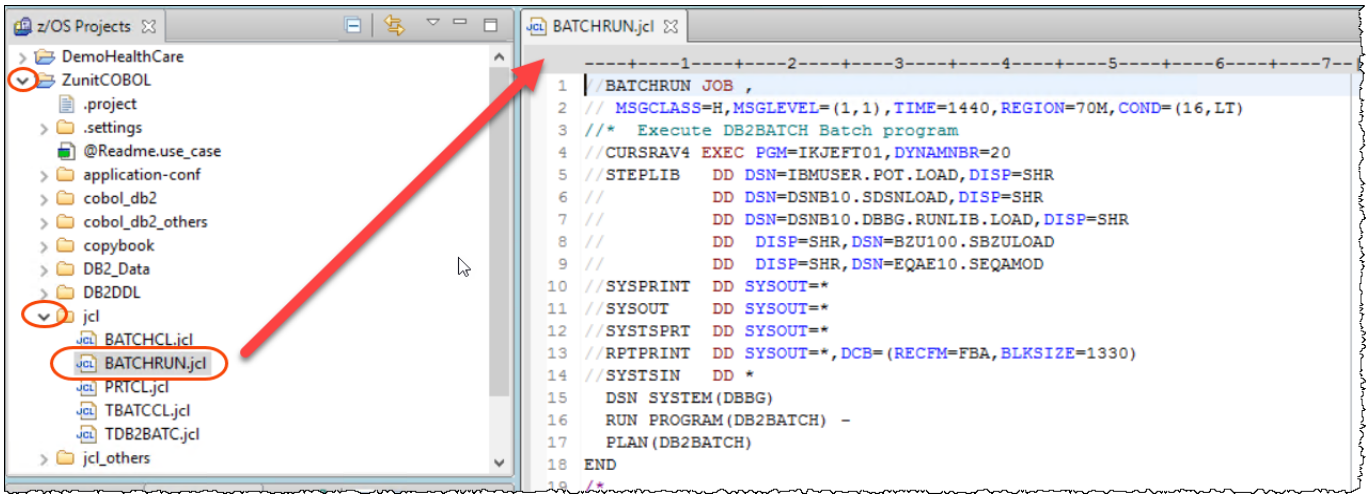
Nothing will happen you are already at this [perspective**.**

1.1.2 On this lab you will use userid **ibmuser** . and password **sys1**.
If you are connected as *ibmuser*, jump to step 1.1.4 otherwise**.**
⏭ Using *Remote Systems* view, right click on **zos.dev** and select **Disconnect**
and then right click on **zos.dev** again and select **Connect**

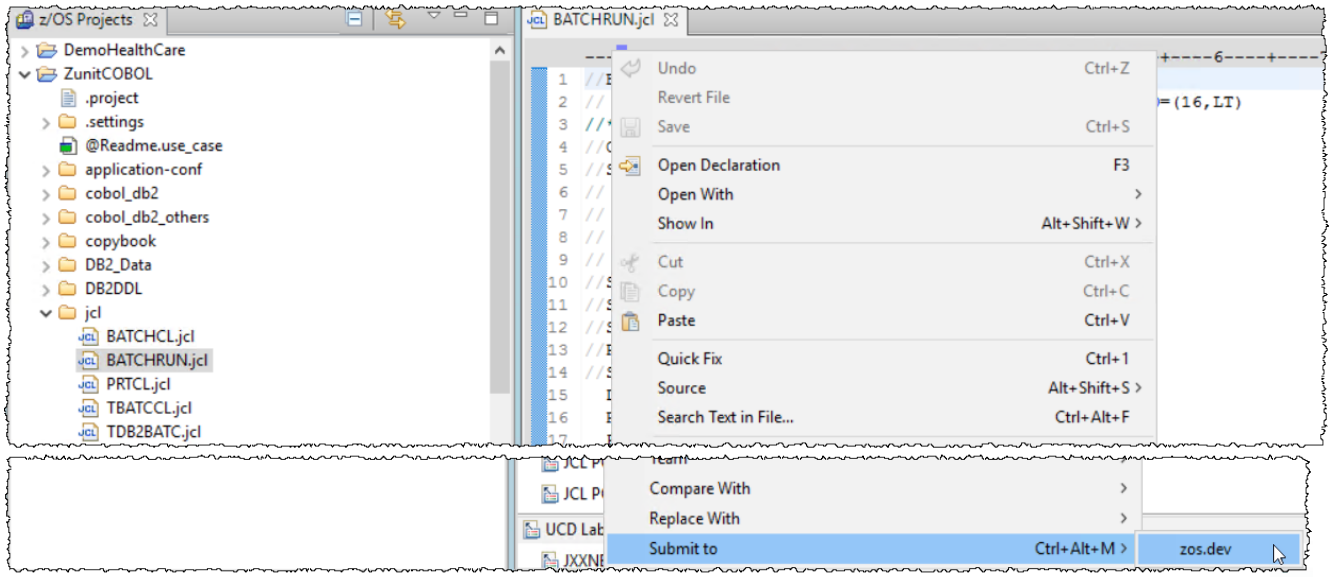
1.1.3 ⏭ Type **ibmuser** as userid and **sys1** as password.
The userid and password can be any case; don't worry about having it in UPPER case.
Click **OK** to connect to z/OS.



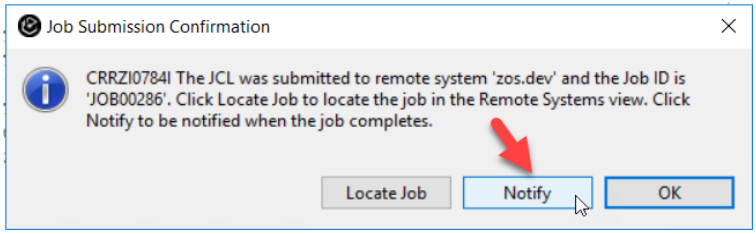
1.1.4 Under **ZunitCOBOL** expand **jcl** and double click on **BATCHRUN.jcl** to edit the JCL that will be submitted for execution.



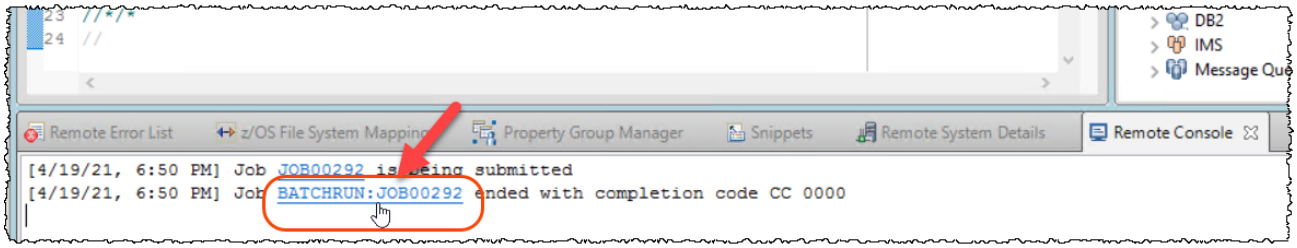
1.1.5 Right click on the JCL edited and select **Submit to > zos.dev**



1.1.6 Click **Notify** to be notified when the execution is complete.

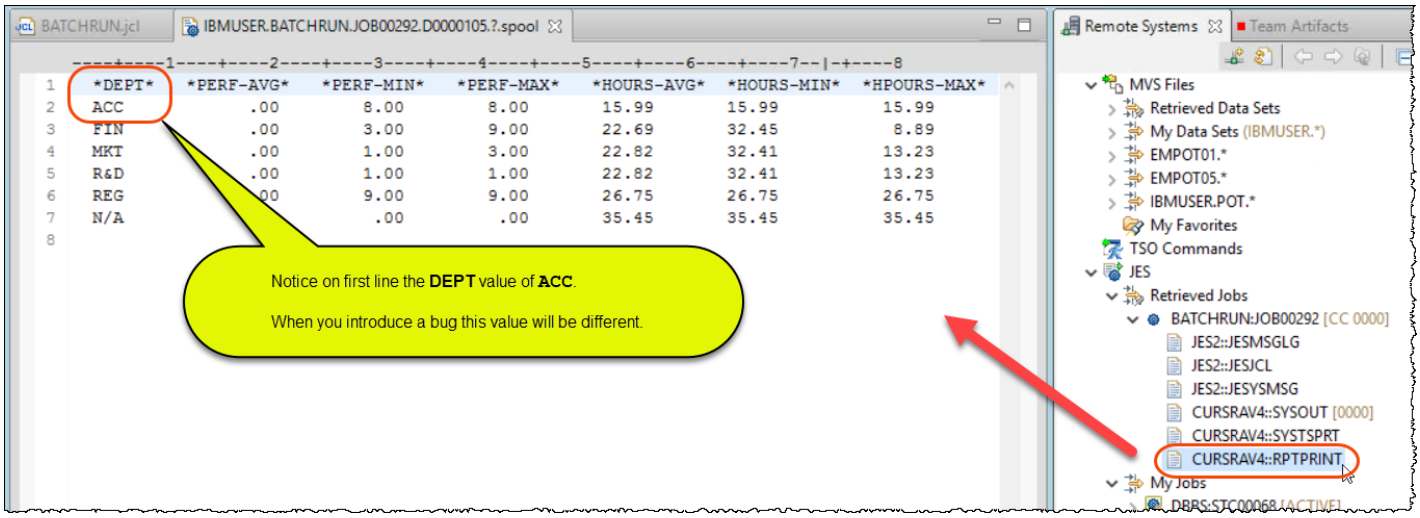


1.1.7 Under *Remote Console*, you will be notified when execution is completed.
⏭ Once the execution ends, **click** on the link **BATCHRUN:JOB00xxx**
(where 00x**xx** is a number) to point to the Job output. This number vary depending on z/OS.

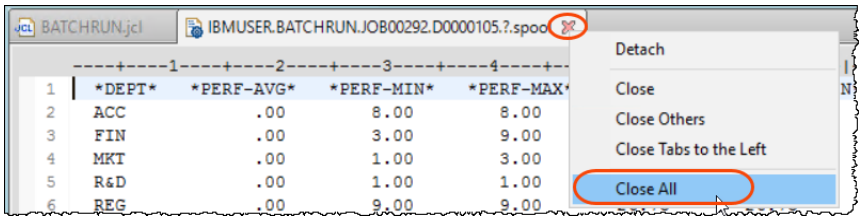


1.1.8 ⏭ Under *Remote Systems* view **scroll down**, expand **JES > Retrieved Jobs > BATCHRUN:JOB00xxx** and **double click CURSRAV4::RPTPRINT** step
and you will see the report produced by the ***DB2PRT*** called COBOL subprogram.

Tip: If there is no jobs under "*Retrieved Jobs"*, is because you did not click on link as stated at 1.1.7.
You can also see this output once you right click on "*My Jobs*" under *JES* and select ***Refresh***.



1.1.9 ⏭ Close all the opened editors using **Ctrl + Shift + F4,**
Or right click on the ⊠. and choose **Close all.**



What have you done so far?

You submitted a JOB to be executed under batch. This job uses two subprograms being invoked. One of the programs invoked dynamically prints a small report from data retrieved from a DB2 table.

Section 2 – Use zUnit to record the batch execution.

Using IDz you will record the COBOL/DB2 batch execution via JCL.

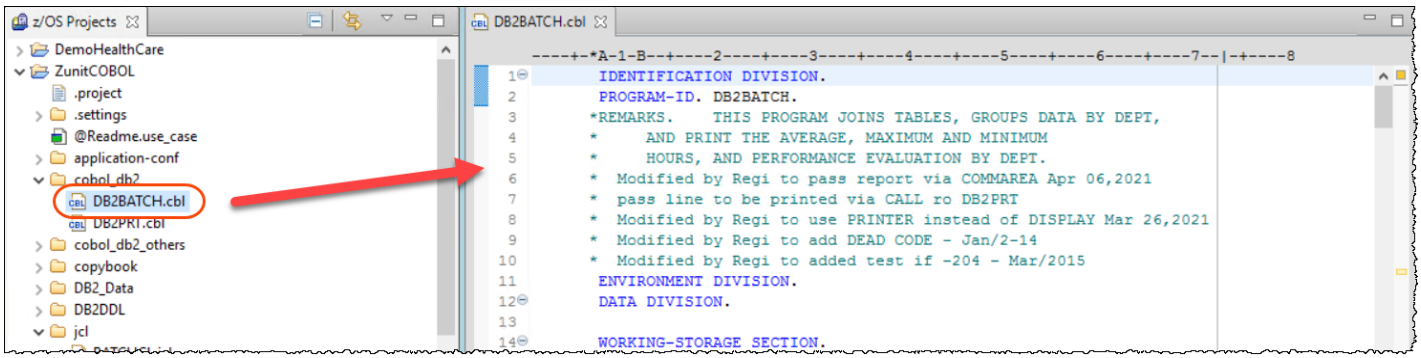
The main COBOL program (**DB2BATCH**) reads from a DB2 table and pass some data to be printed by another dynamically called COBOL subprogram (**DB2PRT**).

Notice that the main COBOL program also invokes a third COBOL program (**REGI0C**) using a static call.

___ 2.1 Understanding the main COBOL program that reads from DB2 table

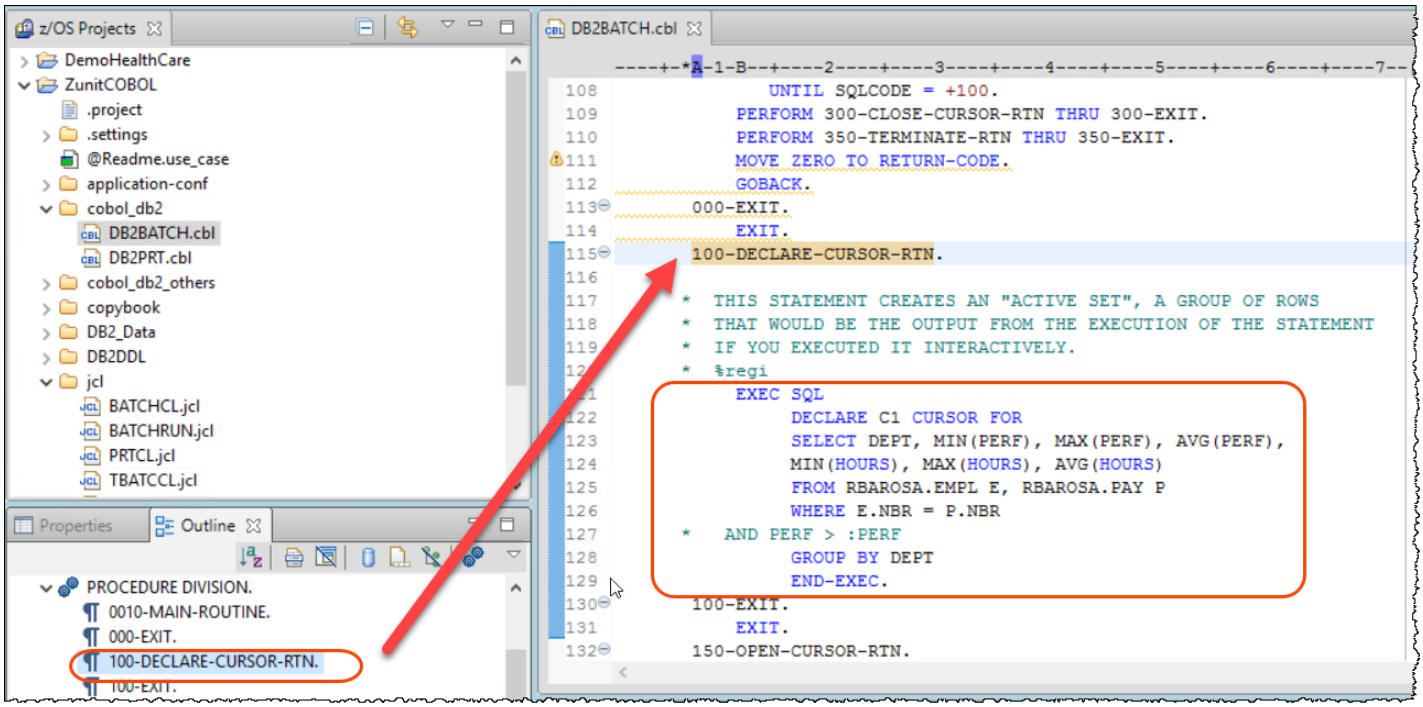
The main COBOL code that reads the DB2 tables is the program ***DB2BATCH***

2.1.1 Using *z/OS Projects* view
double click on **DB2BATCH.cbl** under **ZunitCOBOL/cobol_db2.**
This is the program that you will update later on..



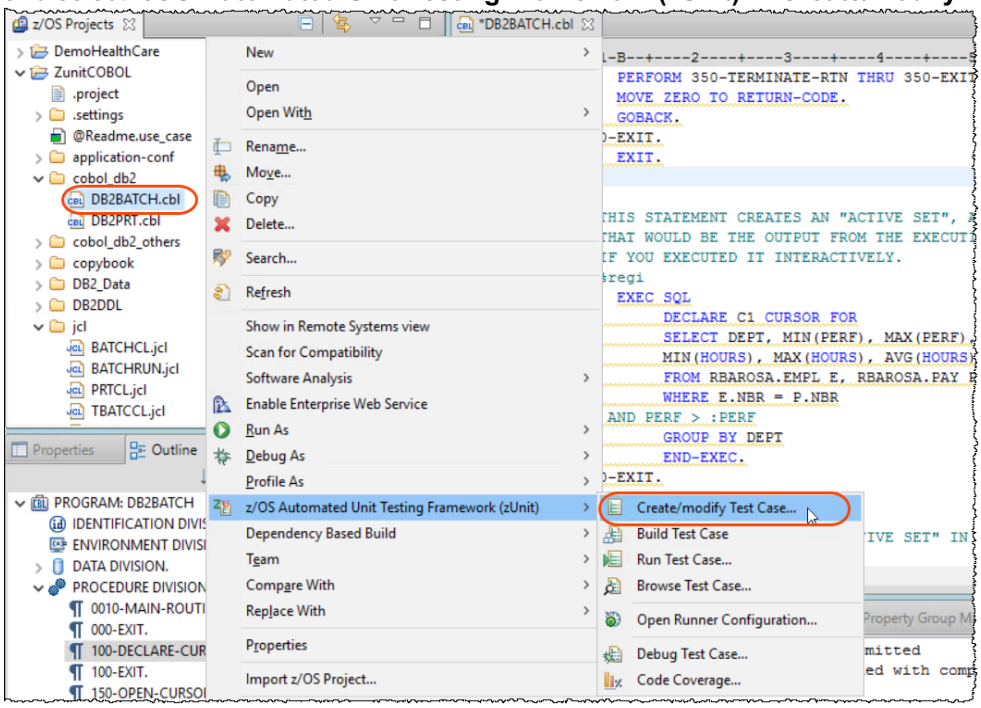
2.1.2 Using the *Outline view* on left expand **PROCEDURE DIVISION** and
click on **100-DECLARE-CURSOR-RTN.**

This is where the DB2 select statement is defined. Notice that the program will execute a DB2 table join and scan the rows resulting from that join. Later on you will introduce a bug in this program.

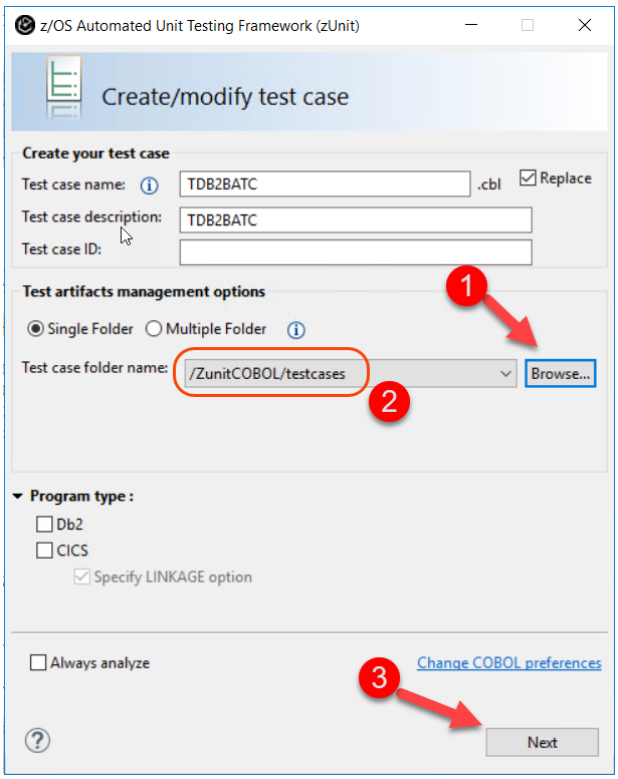

z/OS Projects
DemoHealthCare
ZunitCOBOL
.project
.settings
@Readme.use_case
application-conf
cobol_db2
DB2BATCH.cbl
DB2PRT.cbl
cobol_db2_others
copybook
DB2_Data
DB2DDL
jcl
BATCHCL.jcl
BATCHRUN.jcl
PRTCL.jcl
TBATCCL.jcl
Properties
Outline
PROCEDURE DIVISION.
0010-MAIN-ROUTINE.
000-EXIT.
100-DECLARE-CURSOR-RTN.
100-EXIT.
DB2BATCH.cbl
UNTIL SQLCODE = +100.
PERFORM 300-CLOSE-CURSOR-RTN THRU 300-EXIT.
PERFORM 350-TERMINATE-RTN THRU 350-EXIT.
MOVE ZERO TO RETURN-CODE.
GOBACK.
000-EXIT.
EXIT.
100-DECLARE-CURSOR-RTN.
* THIS STATEMENT CREATES AN "ACTIVE SET", A GROUP OF ROWS
* THAT WOULD BE THE OUTPUT FROM THE EXECUTION OF THE STATEMENT
* IF YOU EXECUTED IT INTERACTIVELY.
* %regi
EXEC SQL
DECLARE C1 CURSOR FOR
SELECT DEPT, MIN(PERF), MAX(PERF), AVG(PERF),
MIN(HOURS), MAX(HOURS), AVG(HOURS)
FROM RBAROSA.EMPL E, RBAROSA.PAY P
WHERE E.NBR = P.NBR
* AND PERF > :PERF
GROUP BY DEPT
END-EXEC.
100-EXIT.
EXIT.
150-OPEN-CURSOR-RTN.

___ 2.2 Recording the batch JCL execution

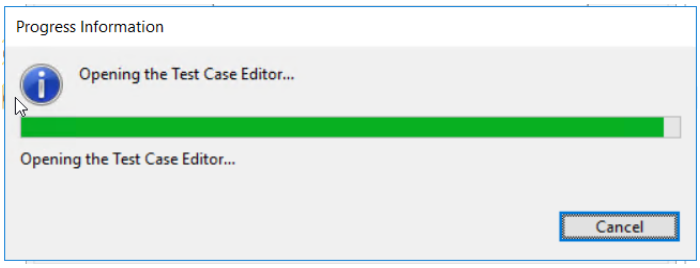
2.2.1 To start the recording, right click on **DB2BATCH.cbl**
and select **z/OS Automated Unit Testing Framework (zUnit)-> Create/modify Test Case..**



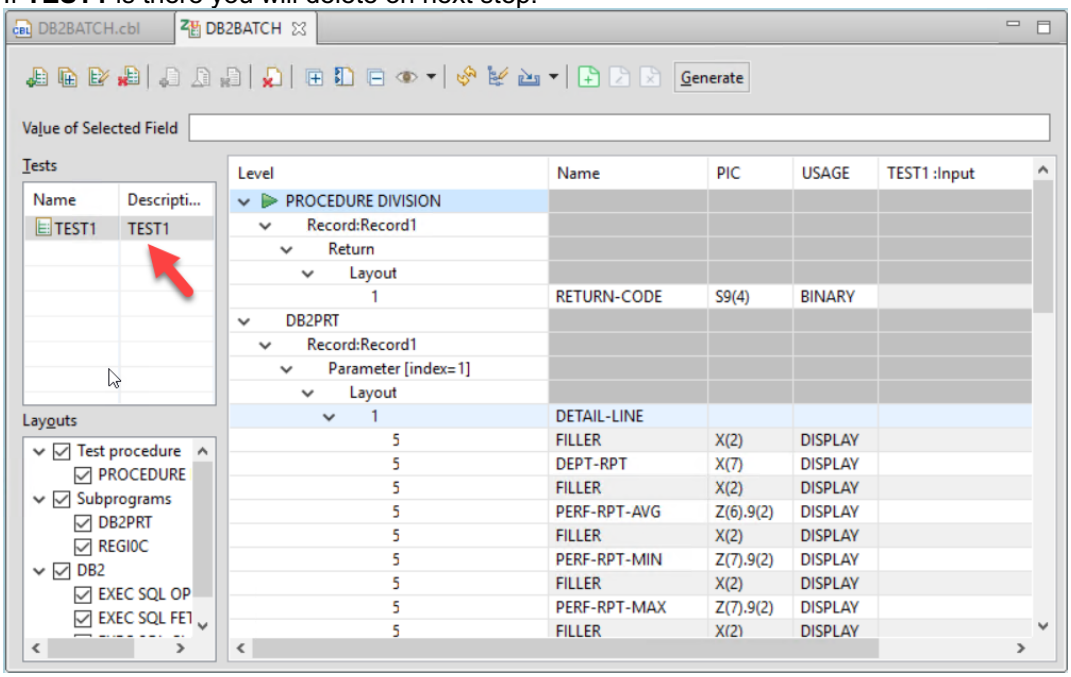
2.2.2 This opens a dialog where you can name your test case.
Using the Browse button select **/ZunitCOBOL/testcases** as folder name and click **Next**.



2.2.3 This operation will generate the test data layout on the local workspace and open the *Test Case Editor.*



2.2.4 The *Test Case Editor*, as shown below. TEST1 may or may not be on your screen. If **TEST1** is there you will delete on next step.



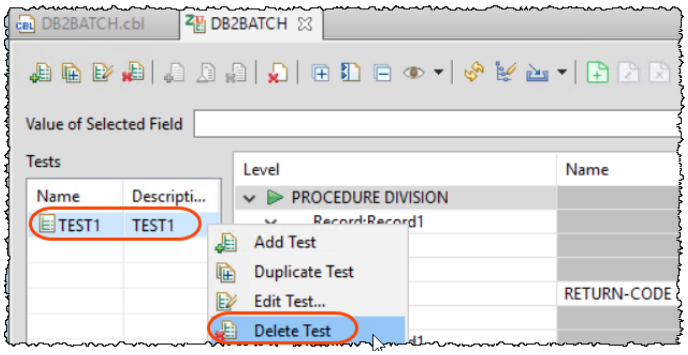
Understanding the *test case editor*

8. The bottom left box summarizes all the input output variable structures – In our exercise, the main COBOL program(*DB2BATCH*) has 2 Subprograms (*DB2PRT* and *REGI0C*) and the LINKAGE Section of those programs are displayed. Also the areas used by DB2 statements are listed.

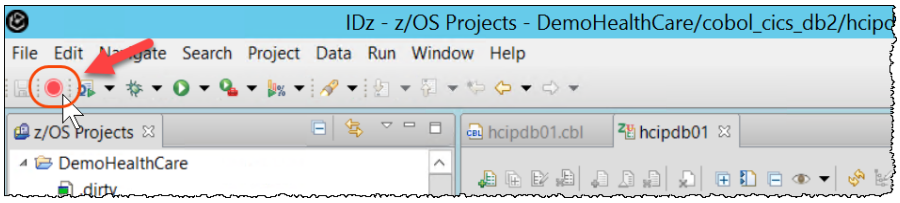
9. The DB2 SQLCA area can be displayed once is selected as below.

The test entry Input and Expected output columns represent the flow of data into and out of the main program, that is, the program being tested by zUnit. When data is added to these columns in a subroutine, the flow of data is:
Input: data passed into the main program: that is, what is passed to the main program when the subroutine completes execution.
Expected output: data passed out of the main program: that is, what is passed to the subroutine when it is called by the main program.

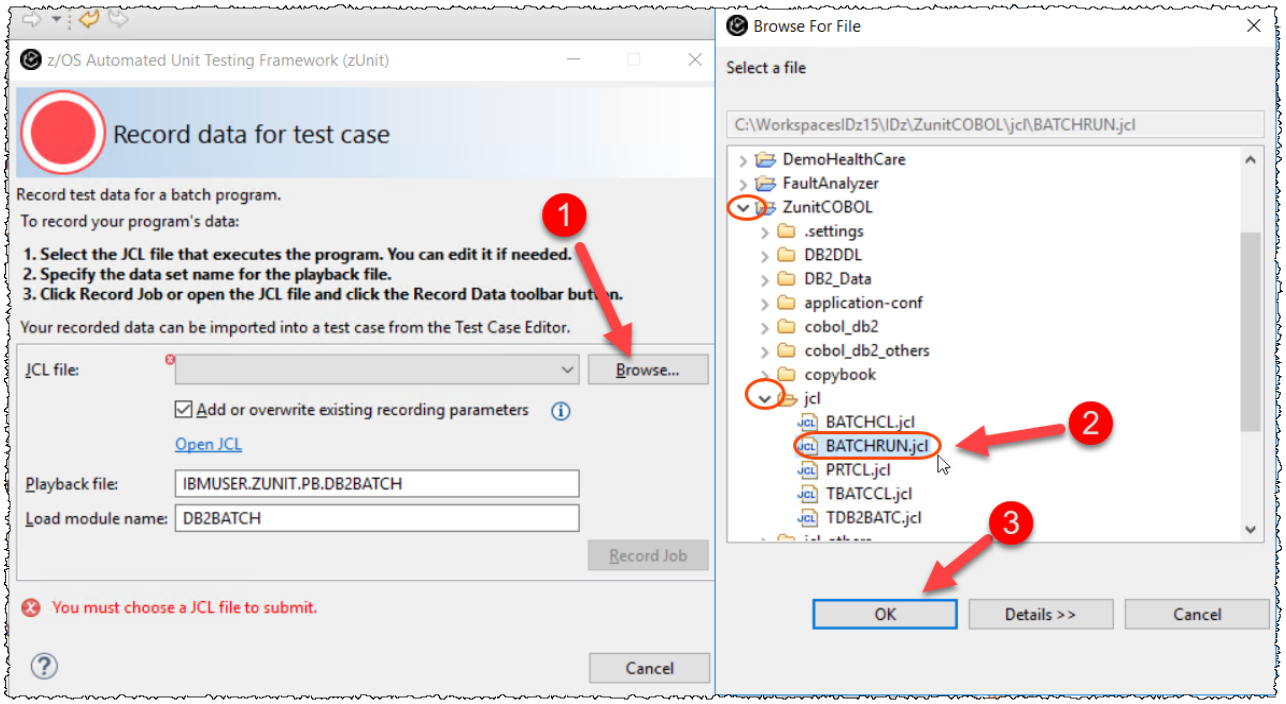
2.2.5 Since we will be recording the program execution, delete the **TEST1** or any other entry (if it exists) by right clicking and selecting **Delete Test**



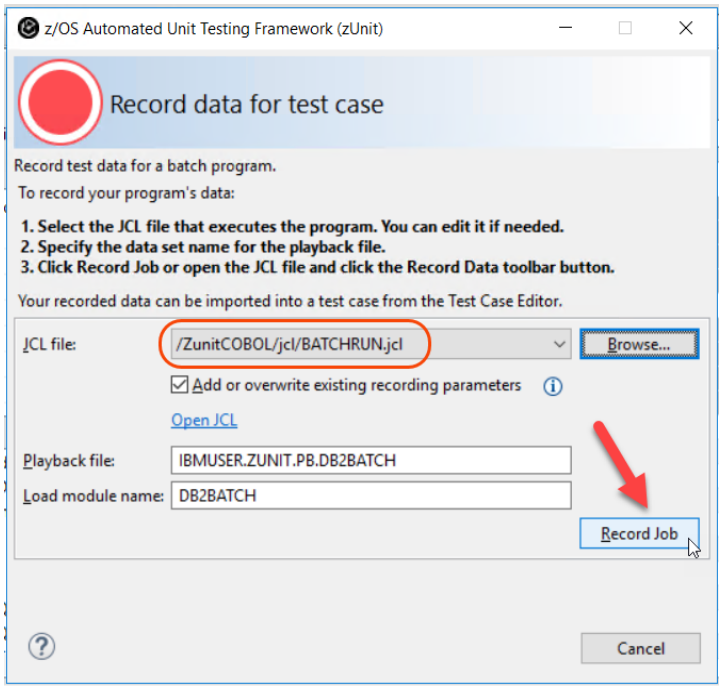
2.2.6 To record from a JCL batch execution into the test case, click the **Record** button on the IDz toolbar.



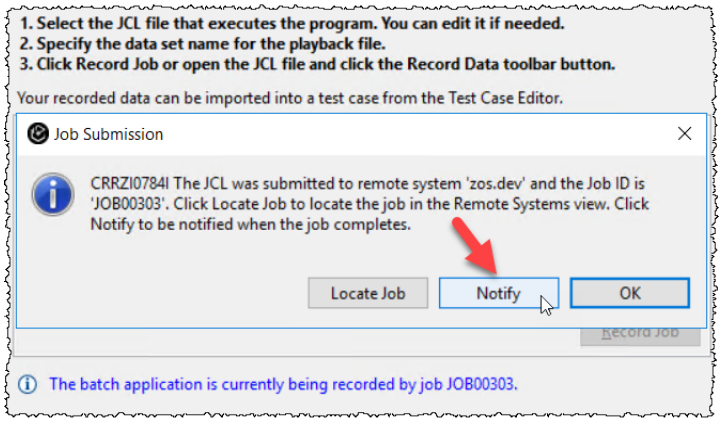
2.2.7 In the dialog that comes up, use the **Browse** button to select **BATCHRUN.jcl** under **ZunitCOBOL/jcl** and click **OK**.



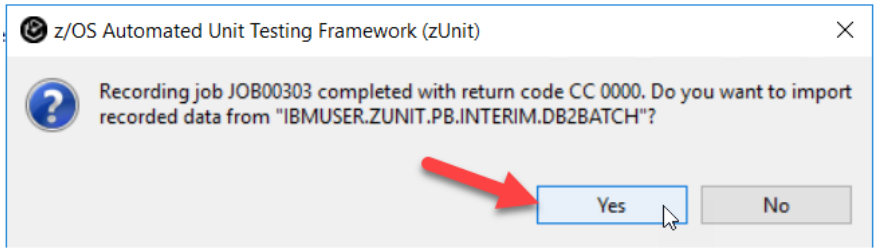
2.2.8 Click on **Record Job** to submit the JCL for execution.



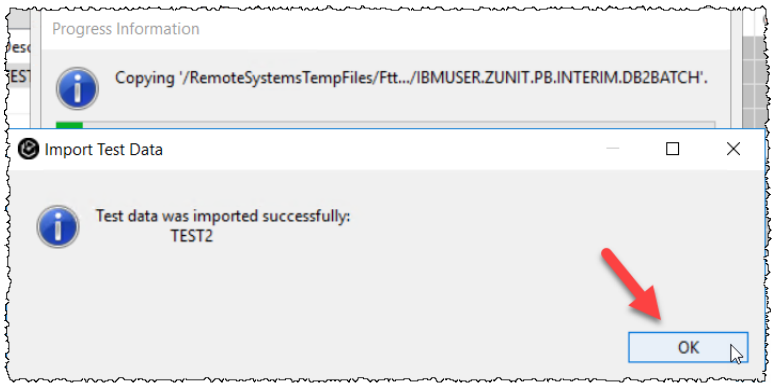
2.2.9 Click **Notify** for the dialog below. The JCL will be submitted to run on z/OS.



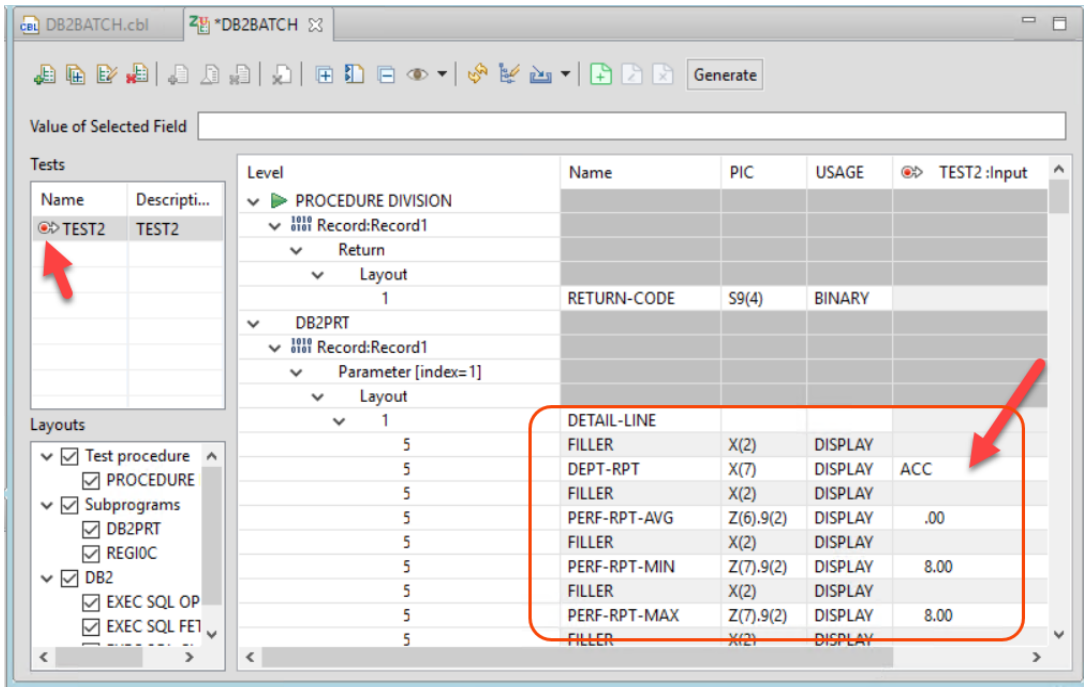
2.2.10 When the job is completed the dialog below is displayed.
Click **Yes** to create the playbackfile and import the test data.



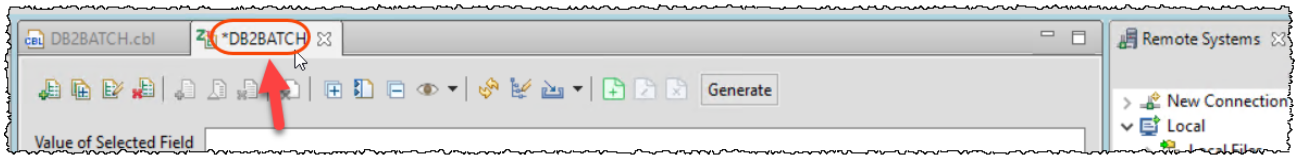
2.2.11 Click **OK** after the data is successfully imported to the test data and playback file.



2.2.12 The test case editor is populated with the data recorded

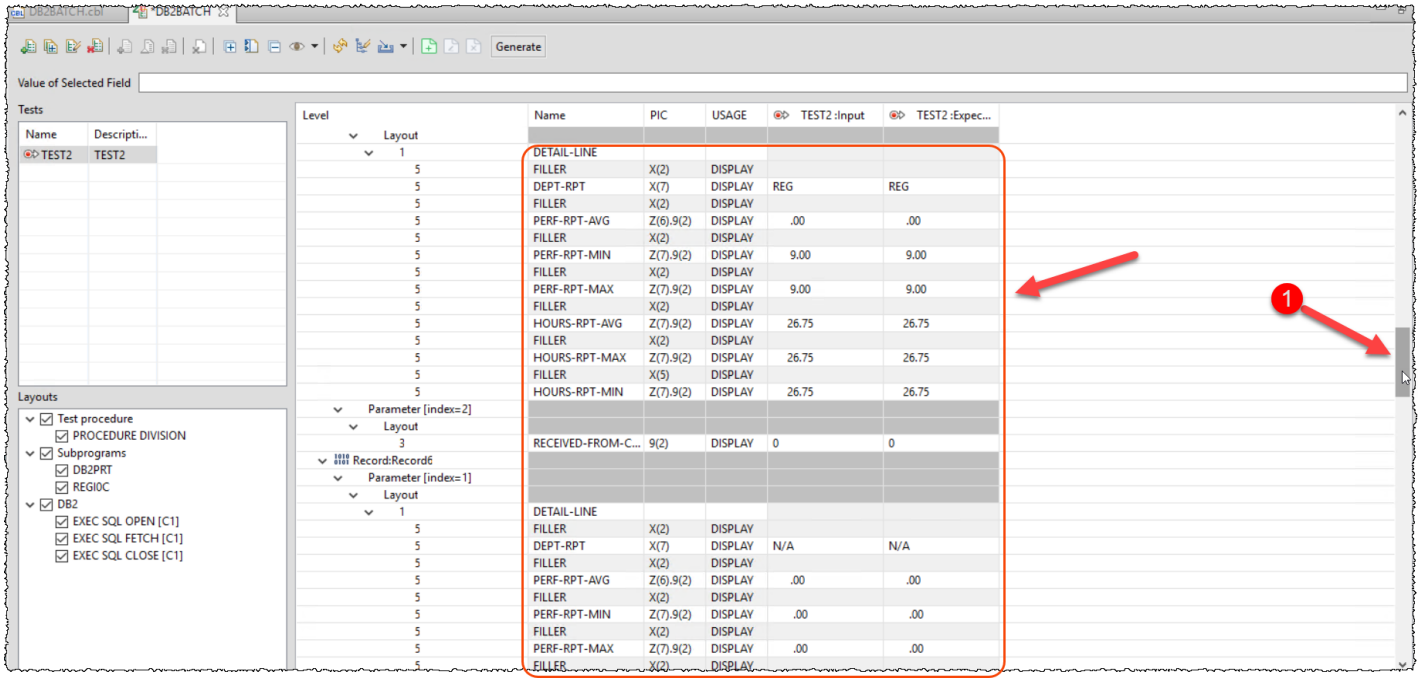


2.2.13 Double click on the **DB2BATCH** title to enlarge the view.



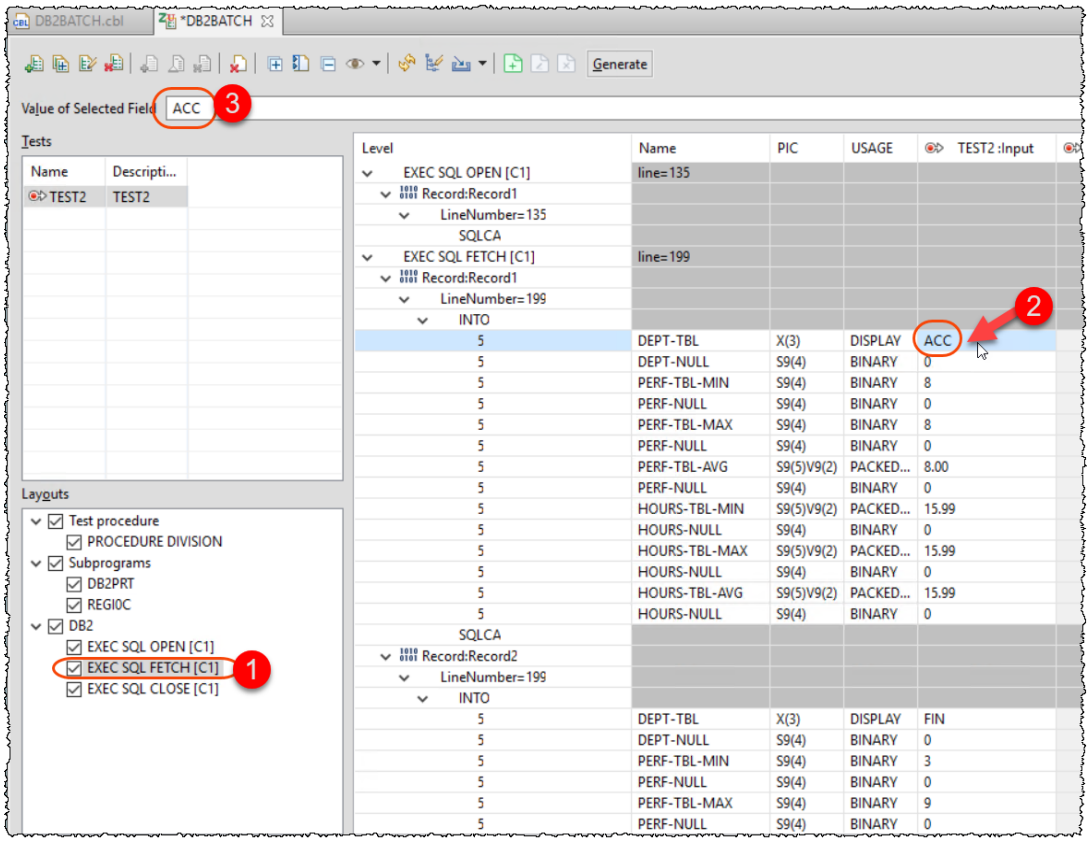
2.2.14 You now see a new test created in the editor and populated with the values from the live run.

Scroll down the editor and notice the 6 rows of data passed to the ***DB2PRT*** subprogram.



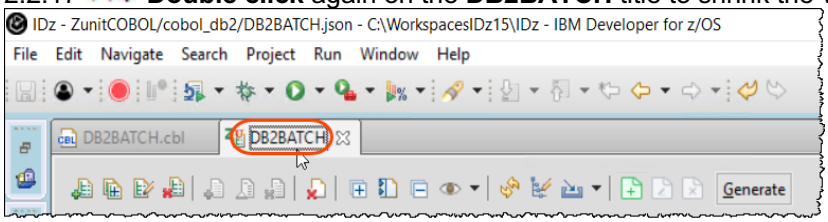
2.2.15 1 Select **EXEC SQL FETCH (C1)** to position the data layout for this statement.

2 Click on **ACC** . and notice that value is displayed on the 3 *Value of Selected Field*



2.2.16 Press **Ctrl+S** to save any changes.

2.2.17 **Double click** again on the **DB2BATCH** title to shrink the view.

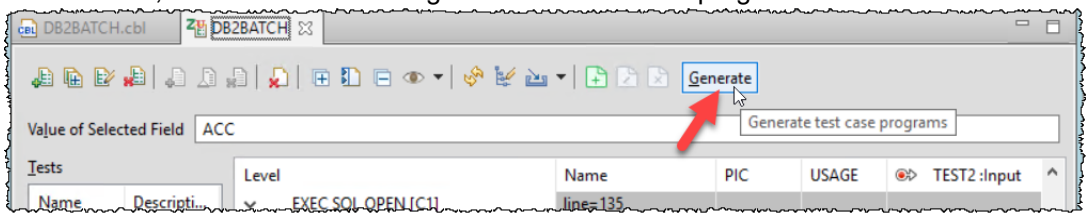


Section 3. Generate, build, and run the unit test generated program.

You will generate, build, and run the unit test for the test case created.

___ 3.1 Generating the COBOL test case programs.

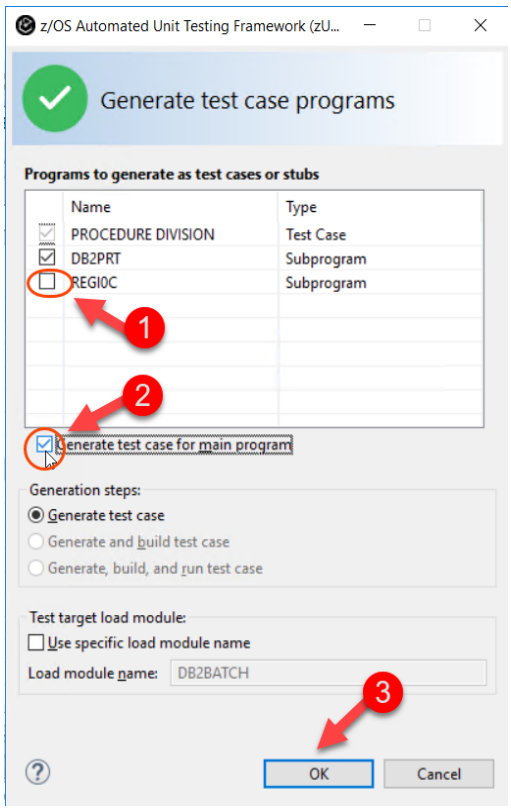
3.1.1 Now that the expected input and output values have been set in the test case editor from the recorded run, **click** on **Generate** to generate the test case programs.



On the ***Generate test case programs*** dialog,

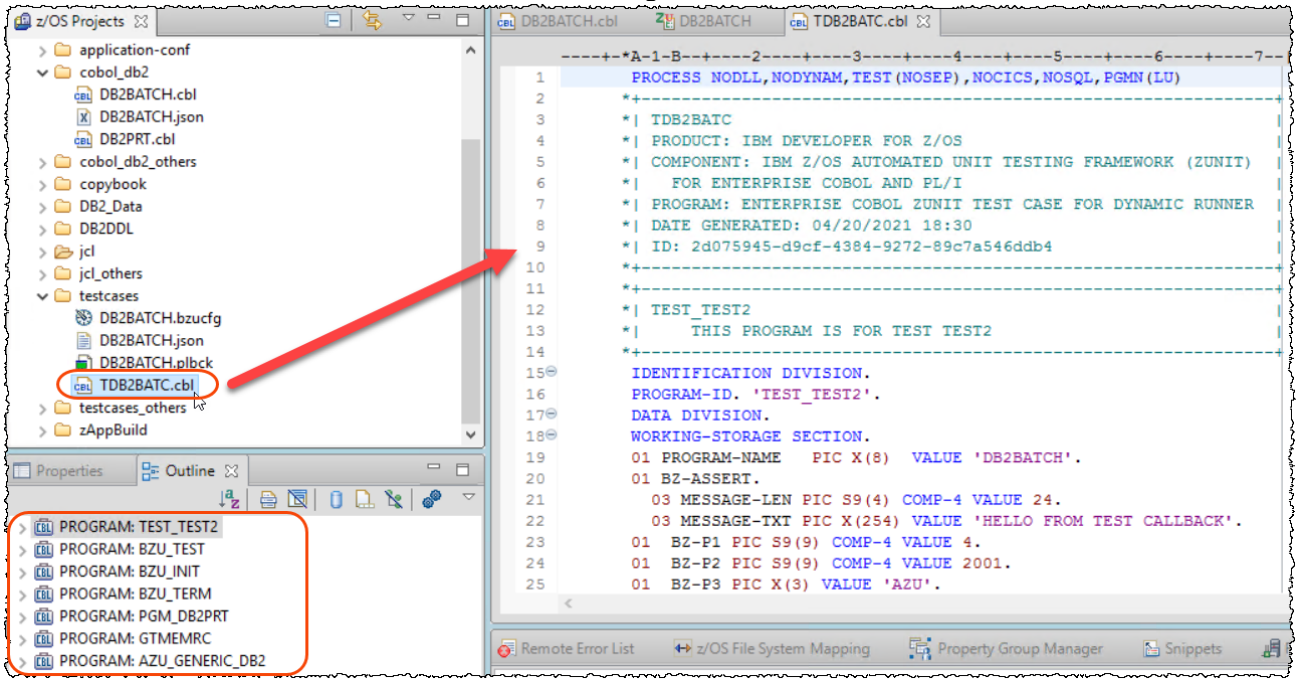
3.1.2 **Un-select REGI0C** (you don't need stubs for this subprogram since is ready)

Select **Generate test case for main program** and then click **OK**.



3.1.3 The COBOL programs that will run the test case are generated under the folder *testcases*.

⏭ Expand **testcases** and double click on **TDB2BATC.cbl** to verify the generated programs.
Notice that in fact 7 COBOL programs were generated from this test case . This can be seen on the ***Outline*** view
You could navigate to each program if time allows, note that NONE of these programs will require CICS or DB2 to be executed. All will be executed in batch using JCL.

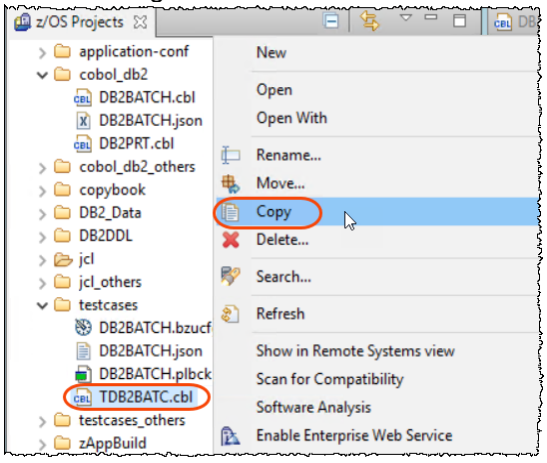


___ 3.2 Generate, build, and run the unit test generated program.

The 7 generated COBOL programs need to be compiled/link edited to be executed. This must be done on z/OS using the COBOL compiler.
To make this easier you could use the IBM DBB (Dependency Based Build) that is part of IDz.
But you also could use the traditional JCL once you moved the generated programs to the z/OS (PDS) to be complied and linked. Here we will show the traditional JCL way.

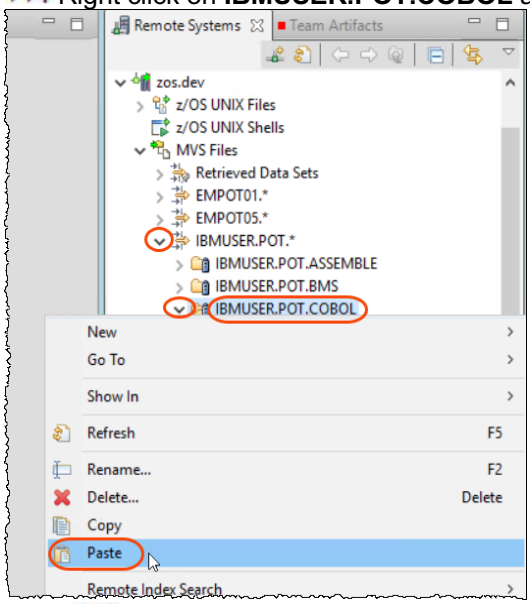
3.2.1 ⏭ Use **Ctrl + Shift + F4** to close all opened editors.

3.2.2 ⏭ Right click on **TDB2BATC.cbl** and select **Copy**

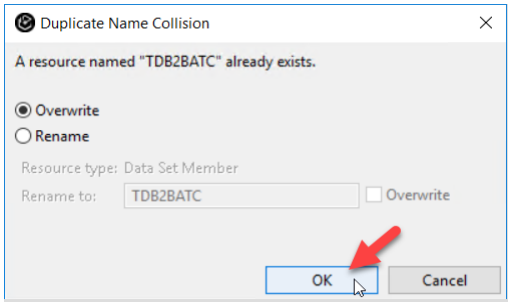


3.2.3 Using the Remote systems (on right), expand **IBMUSER.POT.*** and **IBMUSER.POT.COBOL**

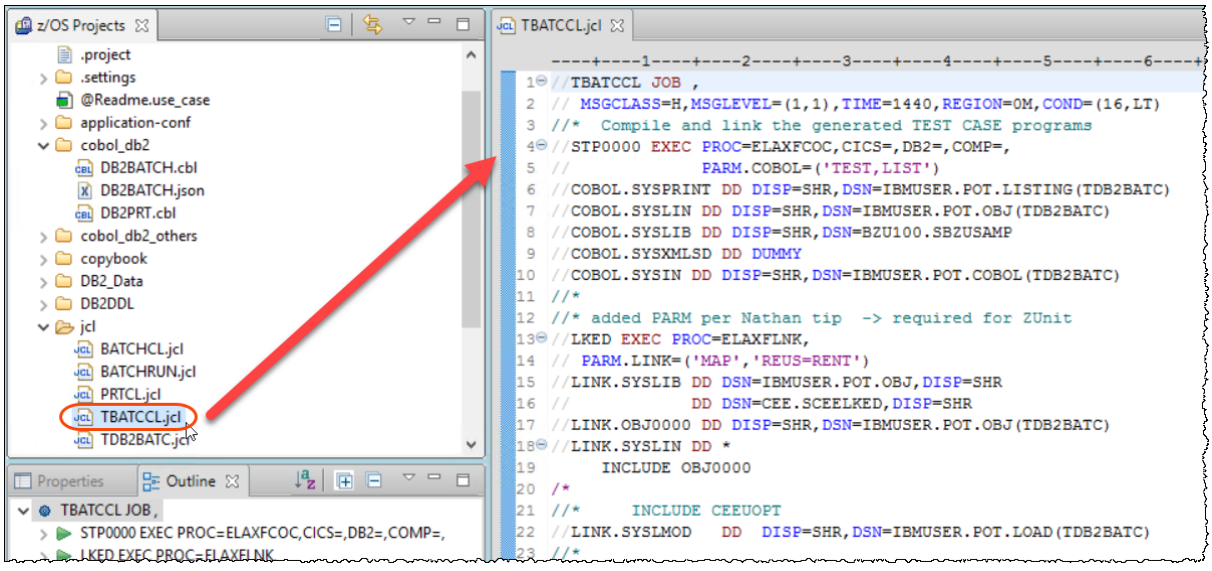
Right click on **IBMUSER.POT.COBOL** and select **Paste**



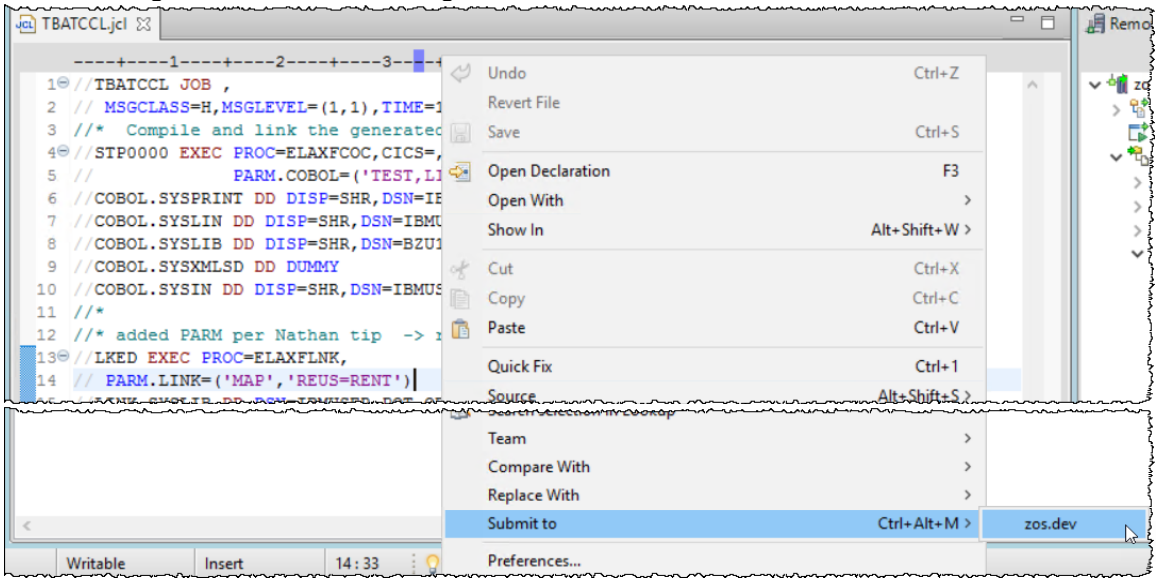
3.2.4 Select **OK** to overwrite this member since it already existed on z/OS



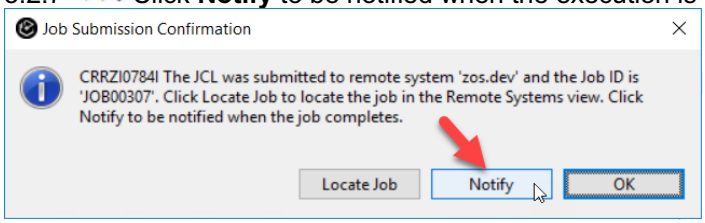
3.2.5 Under z/OS Projects expand **ZunitCOBOL/jcl**, and double click on **TBATCCL.jcl** .
This JCL will compile and link the 7 programs that will create the test case load module.



3.2.6 ⏭ Right click on the JCL being edited and select **Submit to > zos.dev** for execution on z/OS

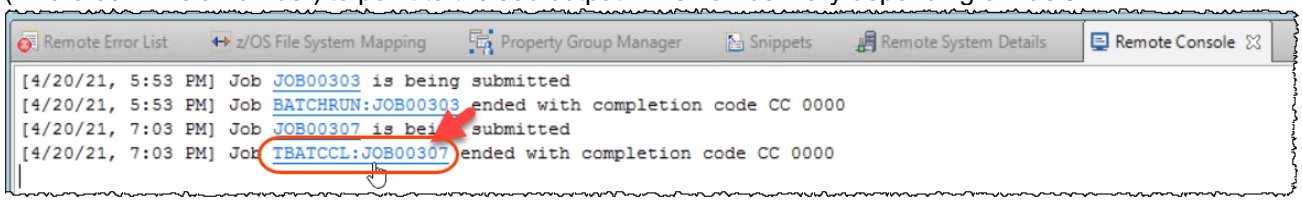


3.2.7 ⏭ Click **Notify** to be notified when the execution is complete.



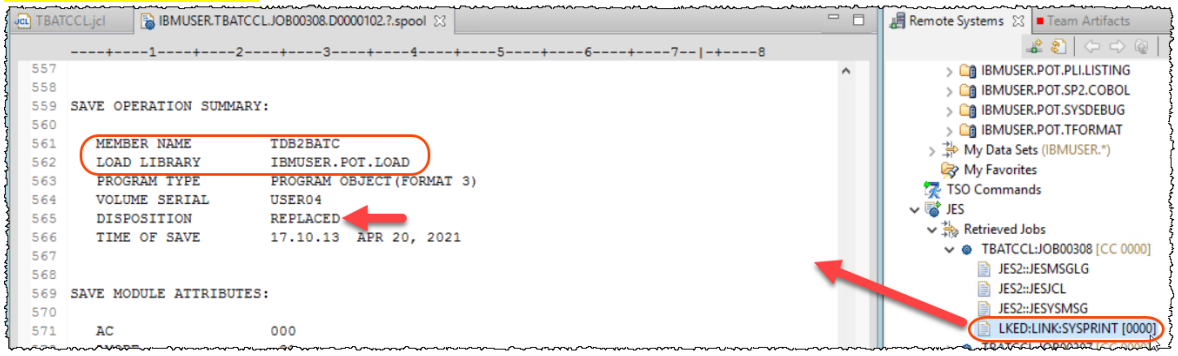
3.2.8 Under *Remote Console*, you will be notified when execution is completed.

⏭ Once the execution ends, **click** on the link **TBATCCL:JOB00xxx**
(where 00x**xx** is a number) to point to the Job output. This number vary depending on z/OS.



3.2.9 ⏭ Under *Remote Systems* view expand **TBATCCL:JOB00xxx** and **double click LKED:LINK:SYSPRINT** step.

⏭ Scroll down the report displayed and you will see the test case load module **TDB2BATC** created at **EMPOT.POT.LOAD** .



3.2.10 Use **Ctrl + Shift + F4** to close all opened editors.

What have you done so far?

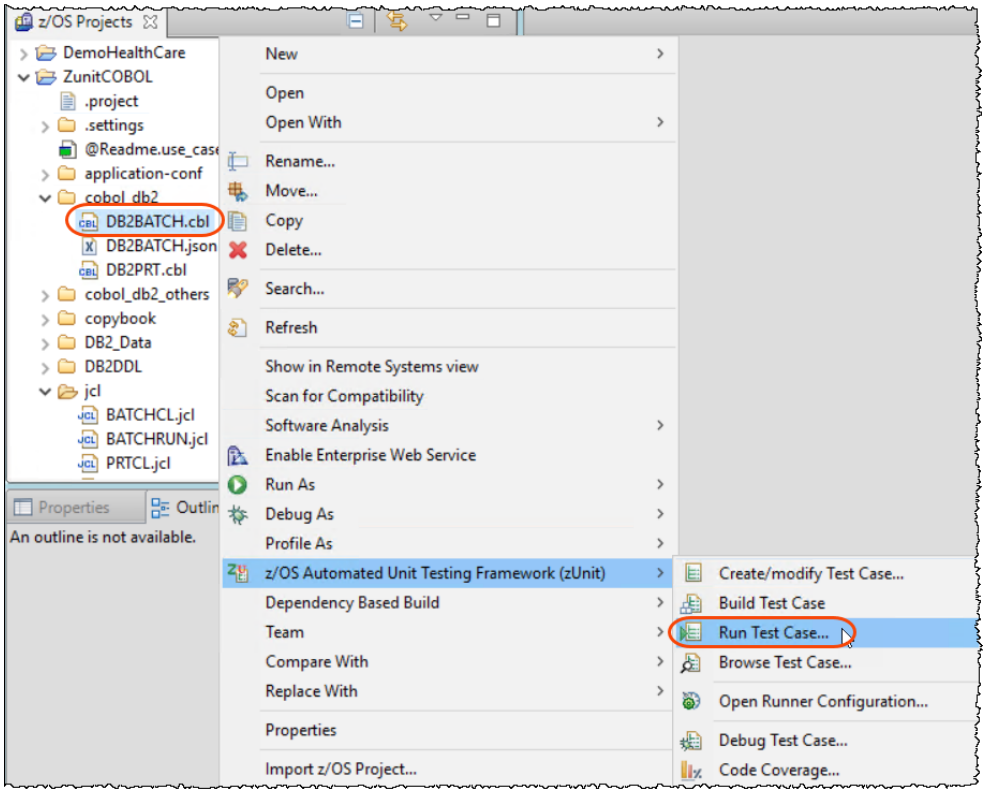
You generated a test case load module (**TDB2BATC**). This load module when executed can verify the correct input and output values handled by the program being verified. The test case can run each time that the program is modified, and the test case must pass it. You will see that

Notice that even that the tested program accessed DB2, this data is stubbed, and the test case will run in Batch via JCL without need to have DB2 active.

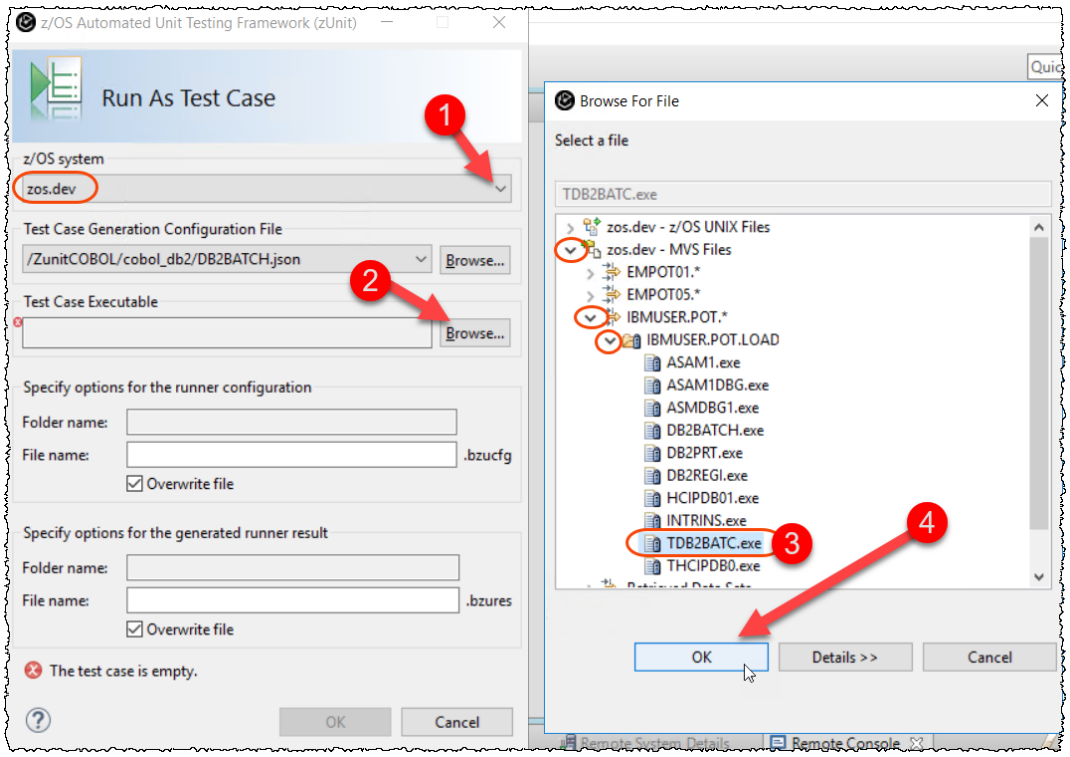
___ 3.3 Running the test case,

Once you have the test cases load module created you can run the test case against the *DB2BATCH* COBOL program. Since you did not make changes to the program the return code must be zero and all tests should pass.

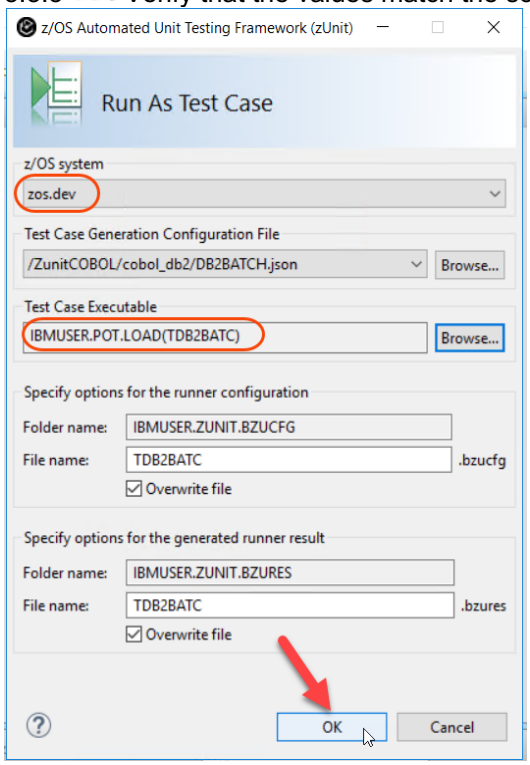
3.3.1 Using *z/OS Projects* view, right click on **DB2BATCH.cbl** and select **z/OS Automated Unit Testing Framework (zUnit)->Run Test Case…**



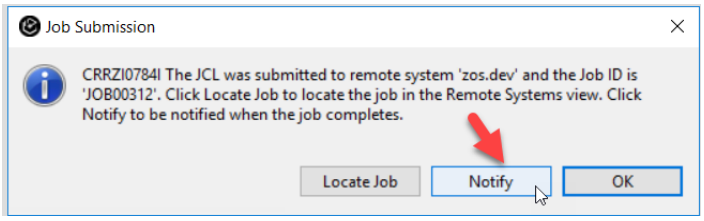
3.3.2 Select **zos.dev** for *z/OS system*,
Use the second **Browse** button to select **`IBMUSER.POT.LOAD(TDB2BATC)`**
under **MVS Files** and **IBMUSER.POT.*.**
Click **OK**.



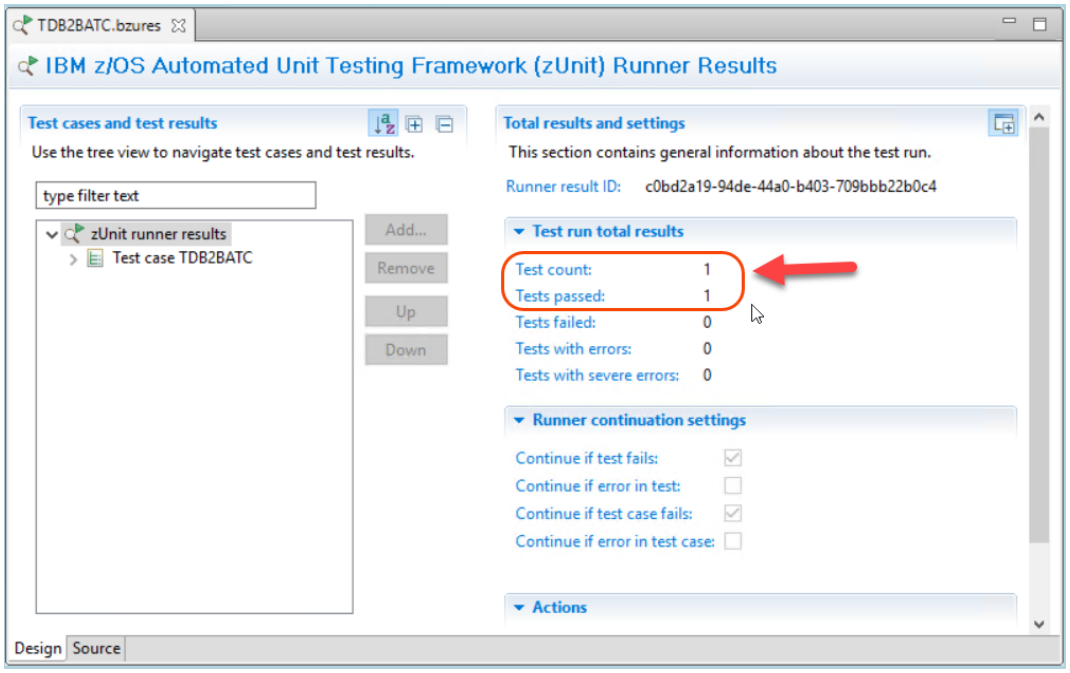
3.3.3 Verify that the values match the screen below and click **OK** to generate and submit a JCL.



3.3.4 A **Job Submission** dialog opens, click **Notify** to be notified of job completion.



3.3.5 Once the unit test run has completed, the results screen is displayed showing test cases ran and passed.



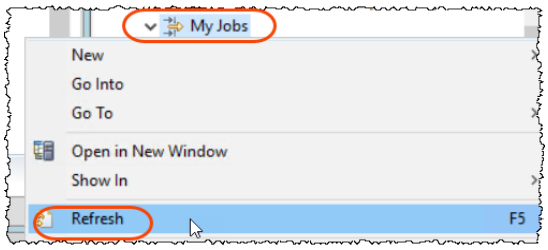
You have now created a test case with data imported from a recorded run and have been successful in testing a COBOL/DB2 batch program without the need of the DB2 environment.

3.3.6 Use **Ctrl + Shift + F4** to close all opened editors.

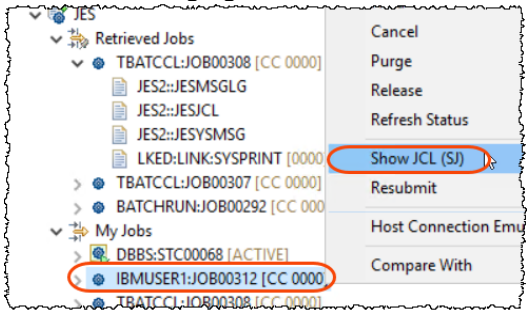
___ 3.4 Verify the JCL submitted that runs the test case

To see the JCL submitted on the previous execution

3.4.1 Using *Remote Systems* view on right, under **JES** right click on **My Jobs**, and select **Refresh**.



3.4.2 Using right click on the last executed and select **Show JCL (SJ)**



3.4.3 The job submitted will be displayed. You could save it in a PDS member and use it when want to run the test cases for this program.
As you see there is no DB2 environment in that batch execution.

Section 4. Modify the COBOL/DB2 program (introduce a bug) and rerun the unit test

Using IDz you will modify the program and introduce a bug. Then you will rerun the unit test created in the previous section.

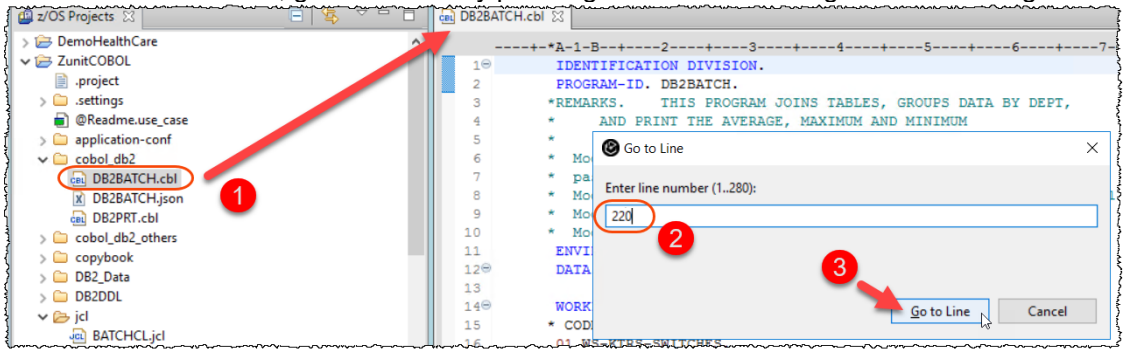
___ 4.1 Modifying the program and introduce a bug

.

4.1.1 On IDz, close all active windows by pressing **Ctrl+Shift+F4**

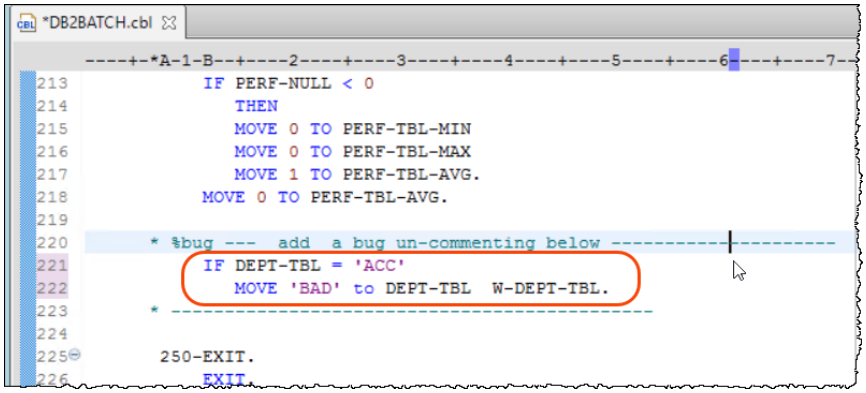
Open **DB2BATCH.cbl** under **cobol_db2** by double clicking on it in the *z/OS Projects* view.

4.1.2 On the editor, go to line 220 by pressing **Ctrl+L** and entering **220** and clicking **Go to Line**.



4.1.3 Change the lines 221 and 222 removing the * from the statement that moves "**BAD**",
Tip -> Could use **Source > Toggle Comment**

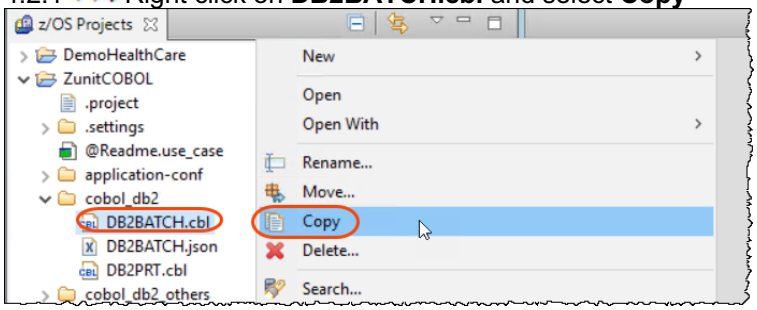
Press **Ctrl+S** to save the changes. and **Ctrl + Shift + F4** to close all editors.



___ 4.2 Re-compile and link the changed program

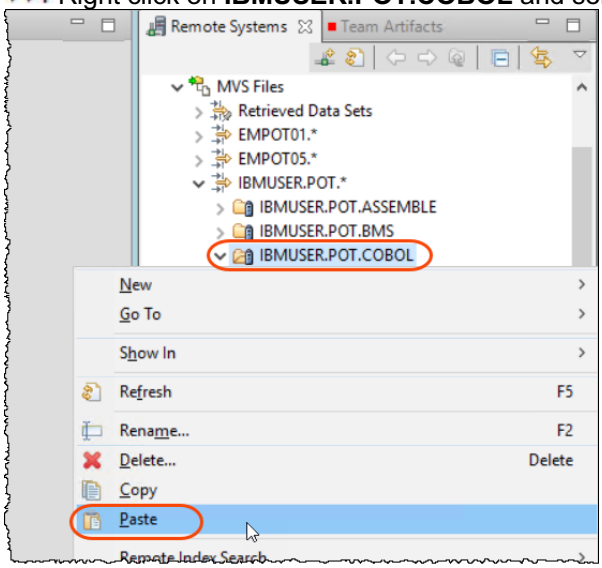
You could use the IBM DBB (part of IDz) to do the compile and link as we already mentioned before, but on this exercise we decided to do it using the old way, via JCL. Notice that the change was made at the local IDz project and the code must be moved to z/OS to compile it there. We had done similar task when we compiled/link the generated test case.

4.2.1 Right click on **DB2BATCH.cbl** and select **Copy**

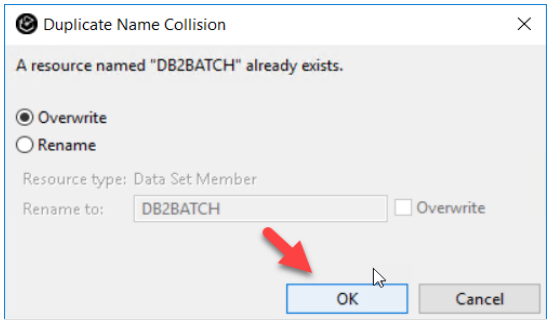


4.2.2 Using the Remote systems (on right),

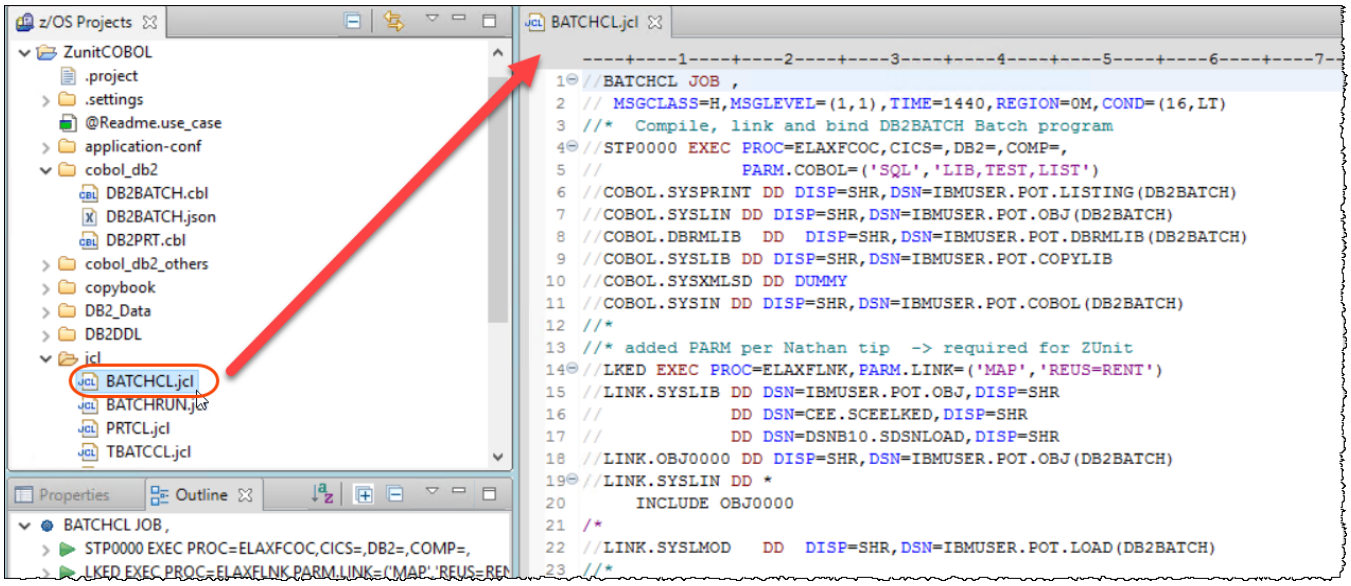
Right click on **IBMUSER.POT.COBOL** and select **Paste**



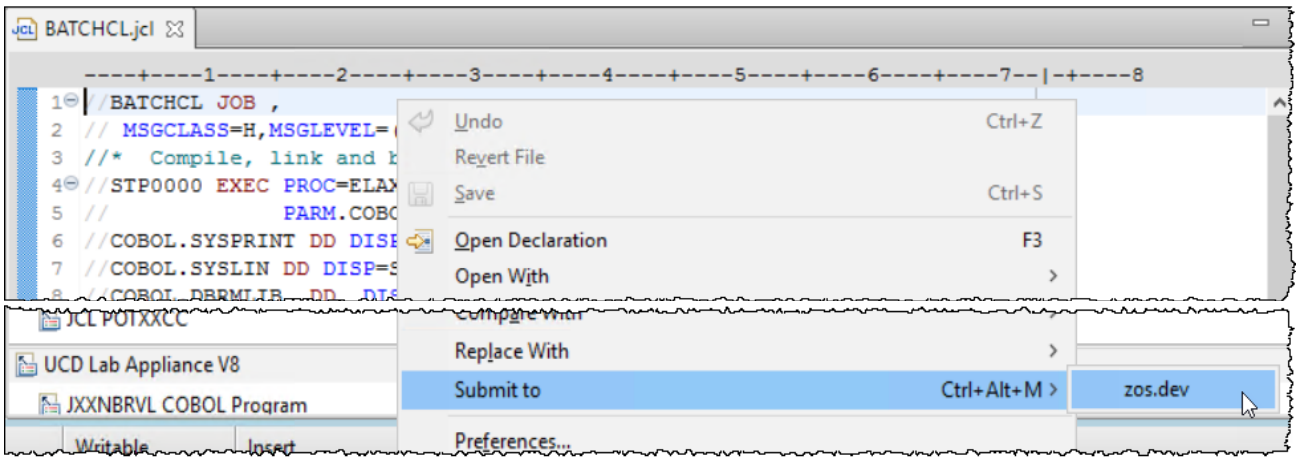
4.2.3 Select **OK** to overwrite this member since it already existed on z/OS



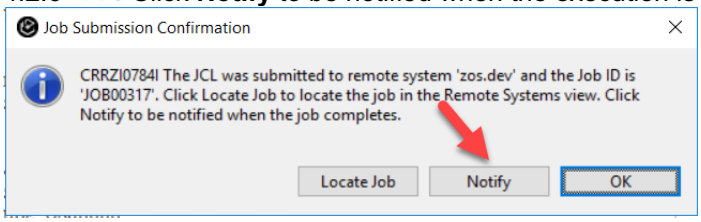
4.2.4 Under z/OS Projects expand **ZunitCOBOL/jcl**, and double click on **BATCHCL.jcl** .
This JCL will compile, link and bind the program being tested.



4.2.5 Right click on the JCL being edited and select **Submit to > zos.dev** for execution on z/OS

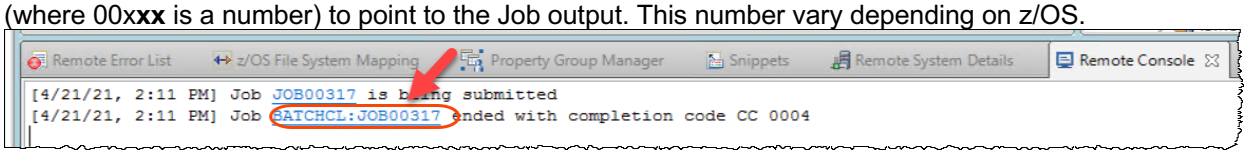


4.2.6 ⏭ Click **Notify** to be notified when the execution is complete.



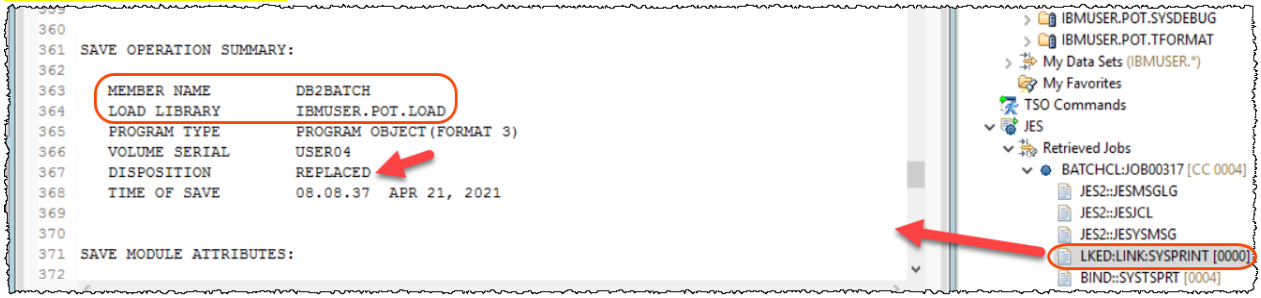
4.2.7 Under *Remote Console*, you will be notified when execution is completed.
The completion code must be 4.

⏭ Once the execution ends, **click** on the link **BATCHJCL:JOB00xxx**
(where 00x**xx** is a number) to point to the Job output. This number vary depending on z/OS.



4.2.8 ⏭ Under *Remote Systems* view expand **BATCHJCL:JOB00xxx** and **double click LKED:LINK:SYSPRINT** step.

⏭ Scroll down the report displayed and you will see the load module **DB2BATCH** is replaced at **IBMUSER.POT.LOAD** .

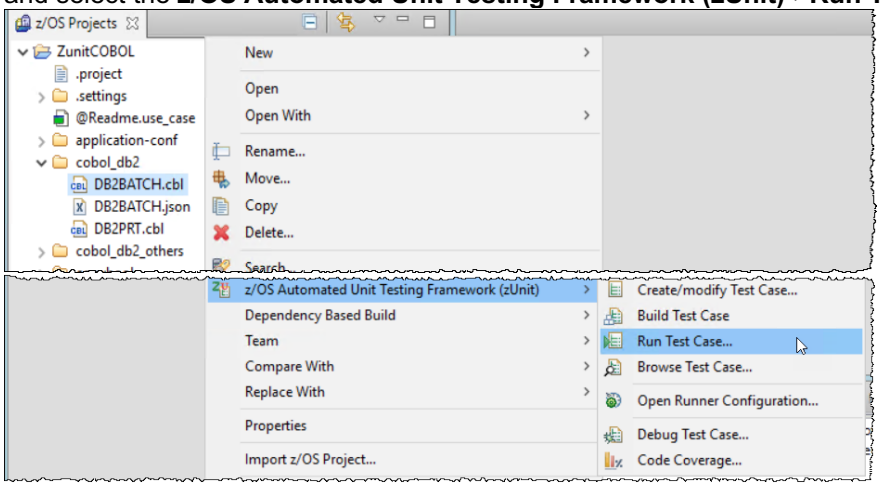


4.2.9 ⏭ Use **Ctrl + Shift + F4** to close all opened editors.

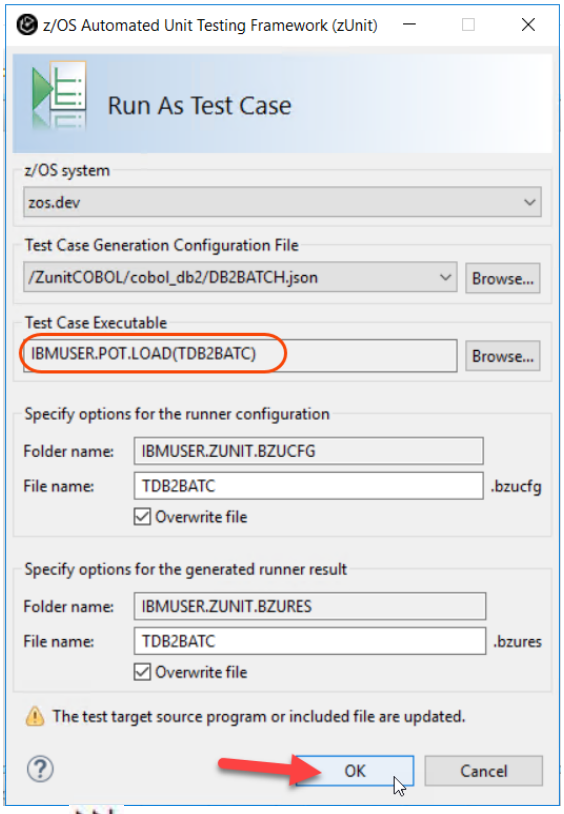
___ 4.3 Running the test case again

Since we introduced a bug, now the test case must fail.

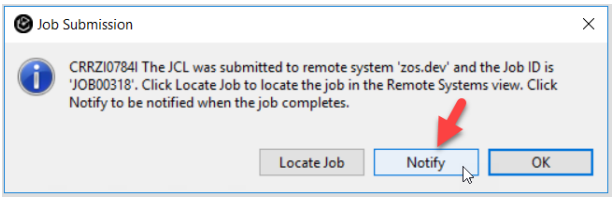
4.3.1 ⏭ On the z/OS Projects view, select **DB2BATCH.cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)->Run Test Case..**



4.3.2 Verify that the test case load module **IBMUSER.POT.LOAD(TDB2BATC)** is already selected and click **OK**

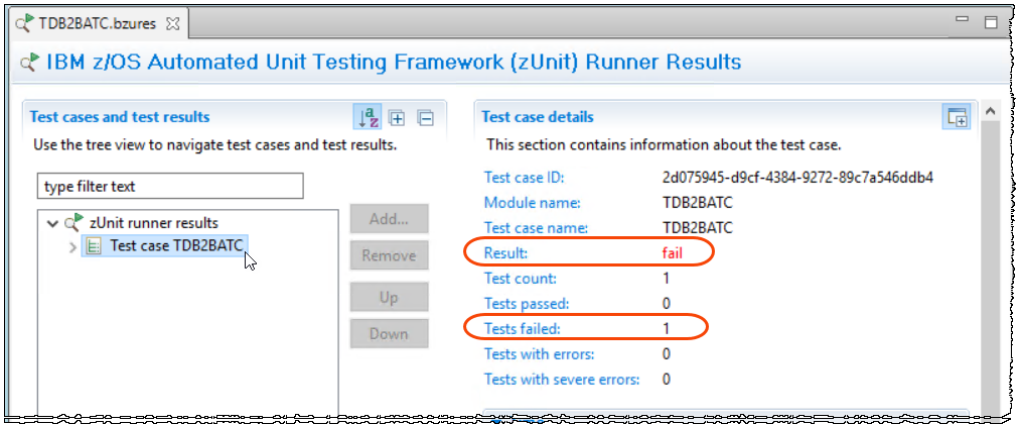


4.3.3 Click **Notify** on the Job Submission Confirmation dialog.

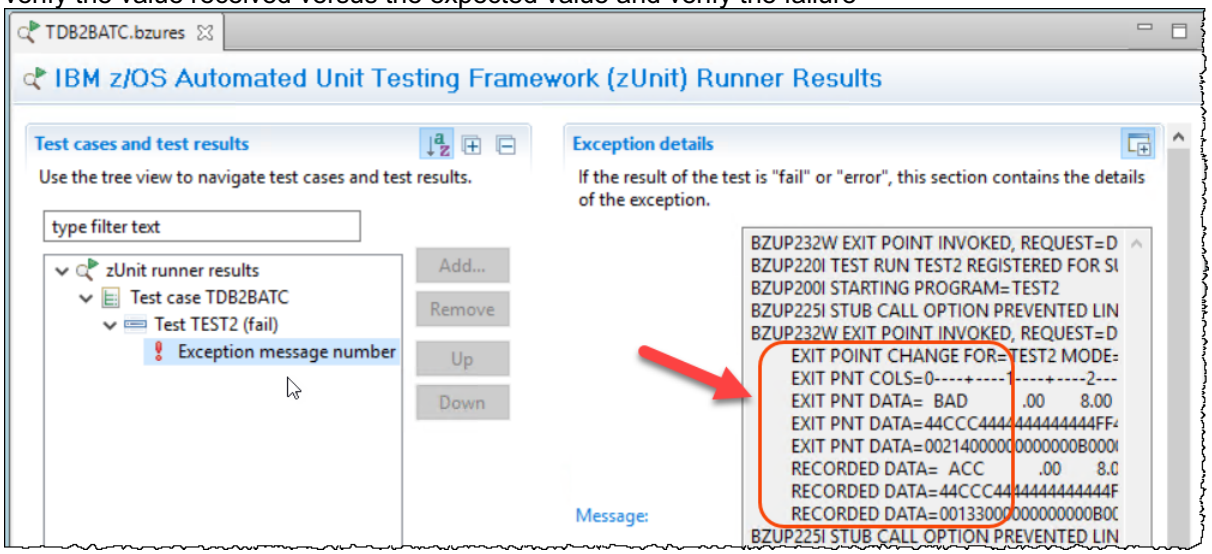


4.3.4 When the test run completes, the test results will be displayed, and it showed that one test ran and it failed. Also notice that the completion code for this JCL execution is 0004 instead of 0000

Click on **Test case TDB2BATC.** The failure is expected because the expected value of the DEPT is different from the actual value.



4.3.5 Expand **Test case TDB2BATC, Test TEST2 (fail)** and click **Exception message number** to verify the value received versus the expected value and verify the failure



As you see the Developer introduced a BUG and the test using ZUnit fails.

PART #2 – Fix program DB2BATCH and re-run the Unit test

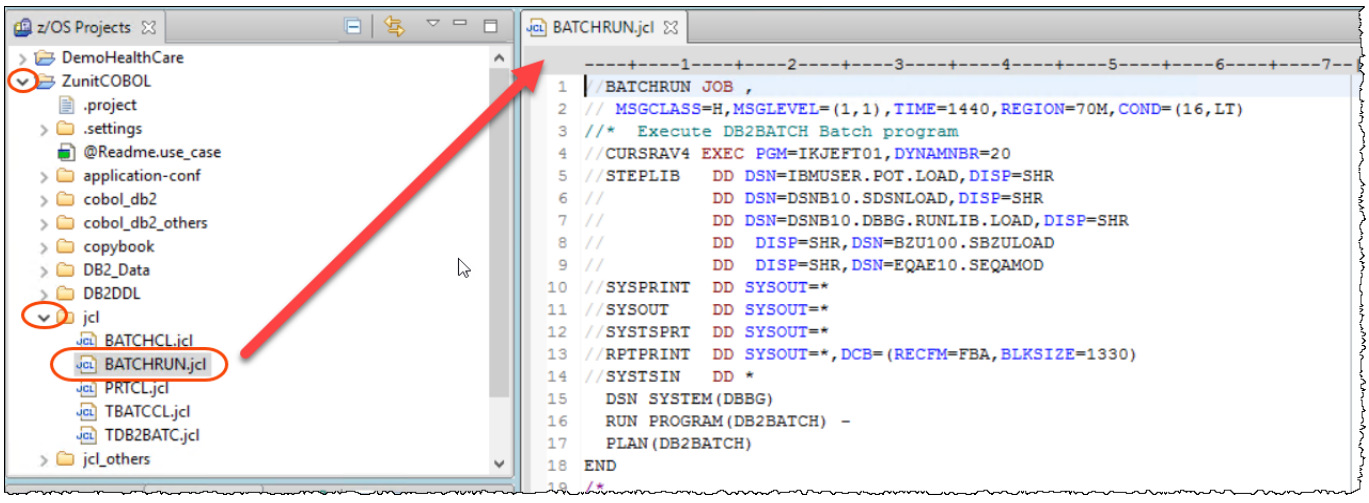
Another developer will see the bug and fix it.
On previous steps you saw on the field *DEPT* the value **BAD** instead of **ACC.**
If you submit the JCL and run this program you will see that bug in the reported printed.

Section 5. Run the batch program and verify the bug .

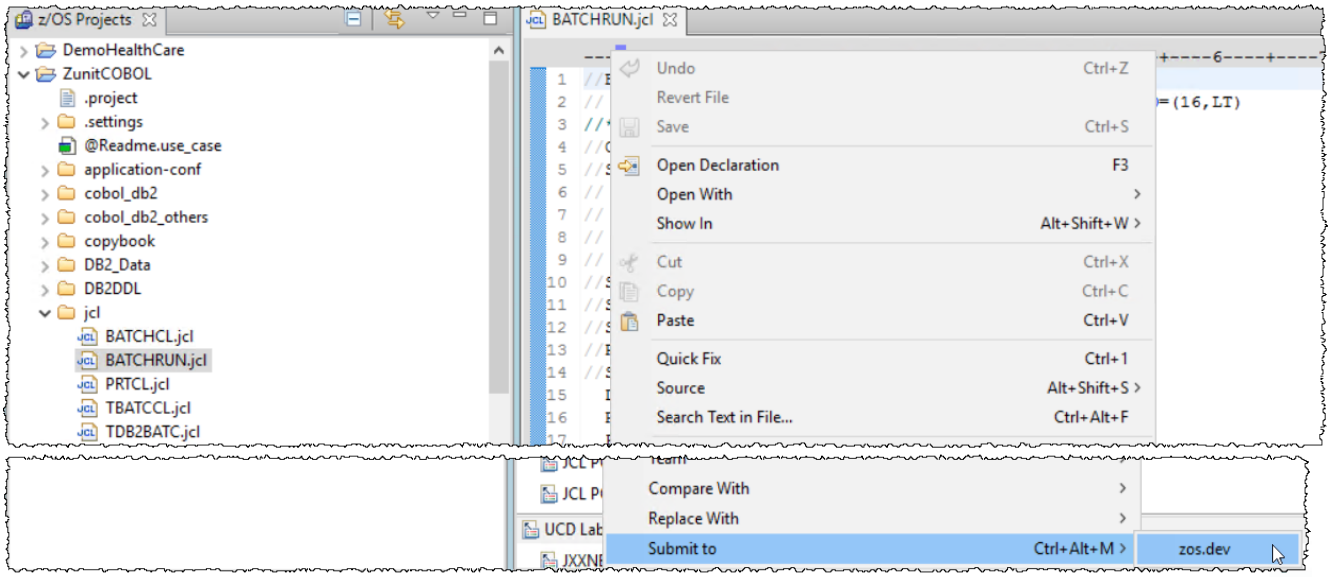
You will run the Batch JCL and observe the bug on the printed report

___ 5.1 Verify the BUG on the printed report

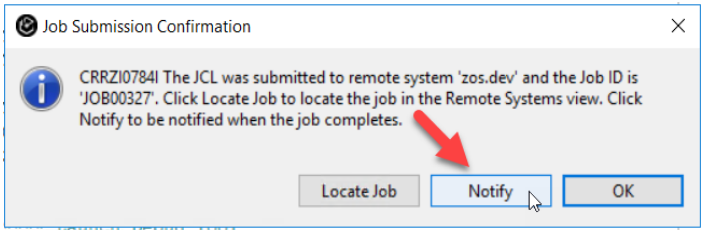
5.1.1 Under **ZunitCOBOL** expand **jcl** and double click on **BATCHRUN.jcl** to edit the JCL that will be submitted for execution.



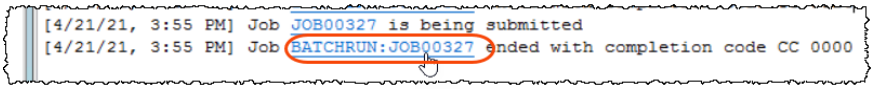
5.1.2 Right click on the JCL edited and select **Submit to > zos.dev**



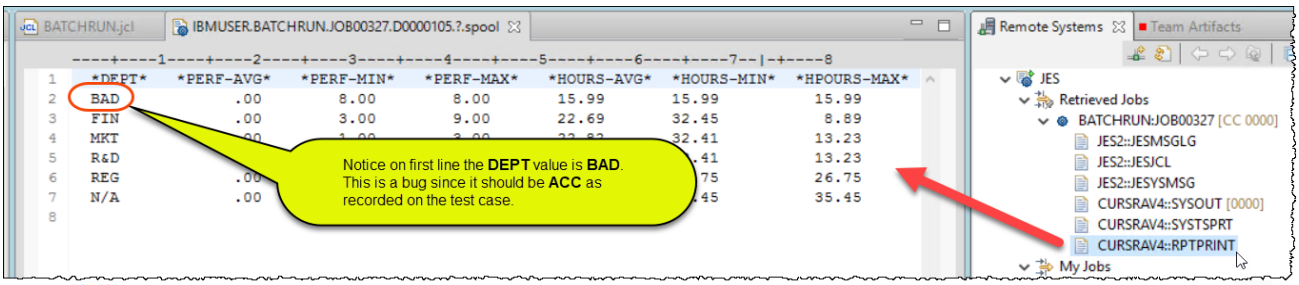
5.1.3 Click **Notify** to be notified when the execution is complete.



5.1.4 Under *Remote Console*, you will be notified when execution is completed.
Once the execution ends, **click** on the link **BATCHRUN:JOB00xxx**
(where 00x**xx** is a number) to point to the Job output. This number vary depending on z/OS.



5.1.5 Under *Remote Systems* view **scroll down**, expand **BATCHRUN:JOB00xxx** and **double click CURSRAV4::IRPTPRINT** step
and you will see the report produced by the ***DB2PRT*** called COBOL subprogram.



5.1.6 Close all the opened editors using **Ctrl + Shift + F4,**

Section 6. Use IDz to fix the bug, recompile the COBOL/DB2 program.

You will fix the using IDz, and rebuilt the executable

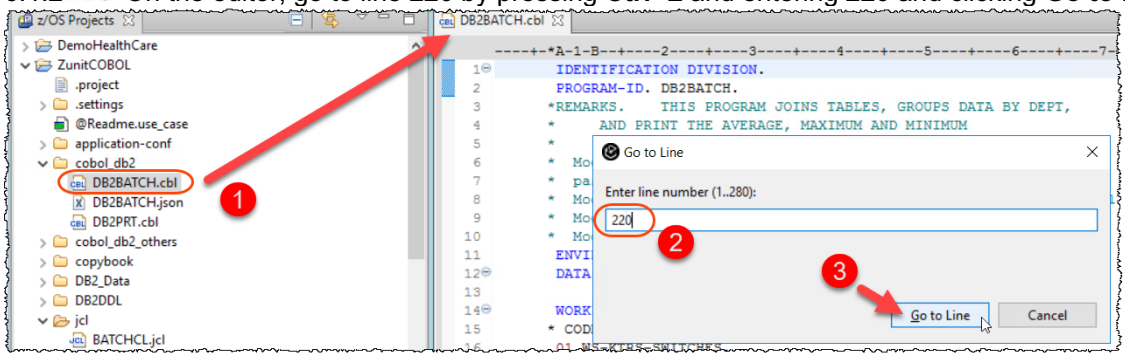
___ 6.1 Modifying the program to eliminate the bug

.

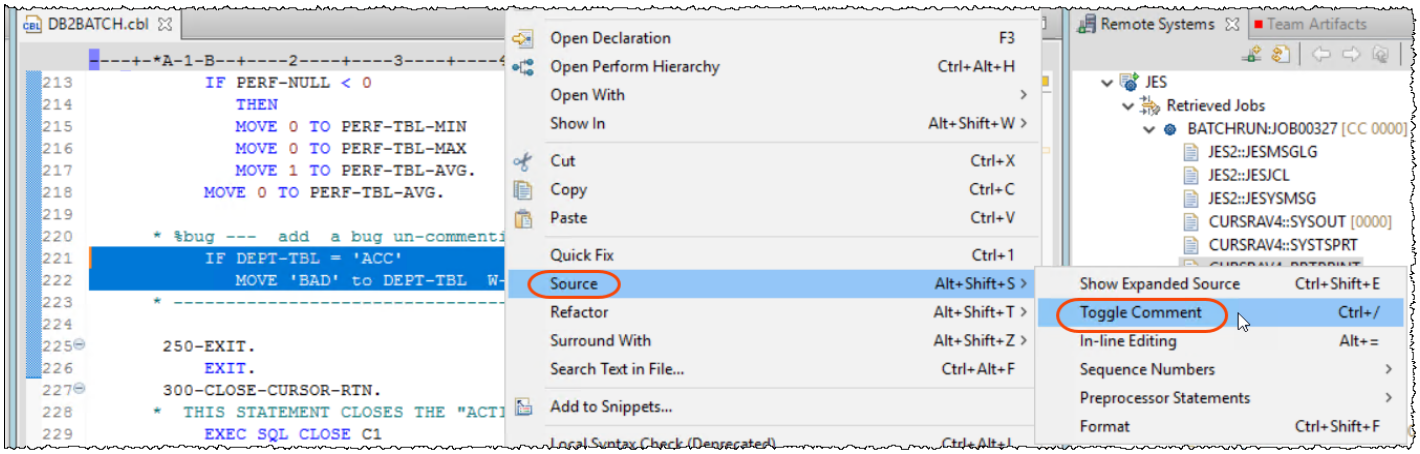
6.1.1 Using *z/OS Projects* view open **DB2BATCH.cbl** under **cobol_db2** by double clicking on it

.

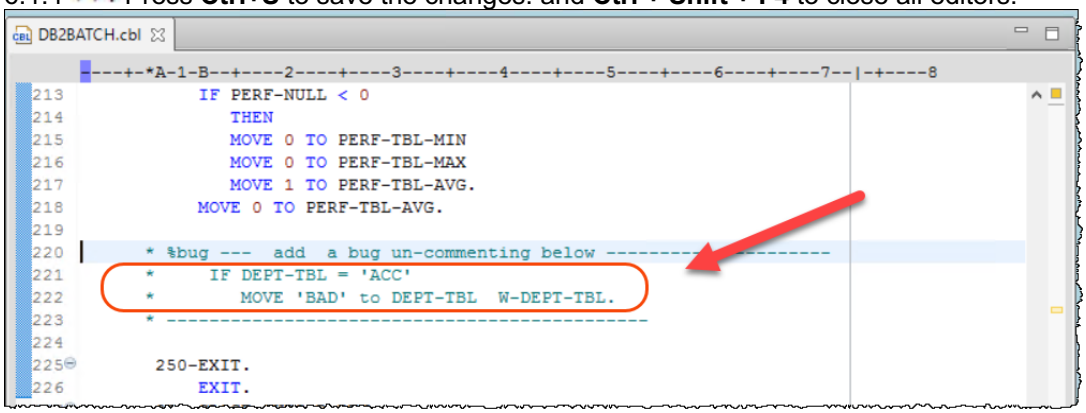
6.1.2 On the editor, go to line 220 by pressing **Ctrl+L** and entering **220** and clicking **Go to Line**.



6.1.3 Change the lines 221 and 222 adding an * on the column 7.
The easiest way to do that is **selecting those 2 lines**, right click and select **Source > Toggle Comment**



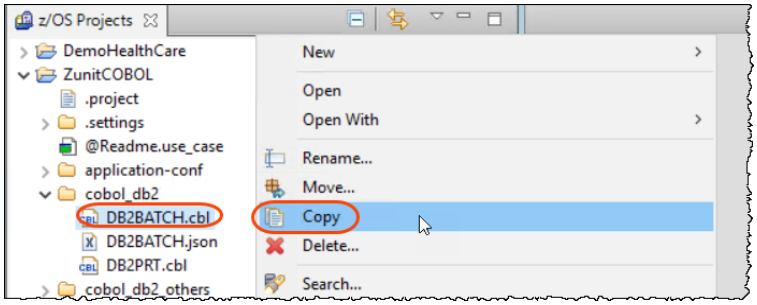
6.1.4 Press **Ctrl+S** to save the changes. and **Ctrl + Shift + F4** to close all editors.



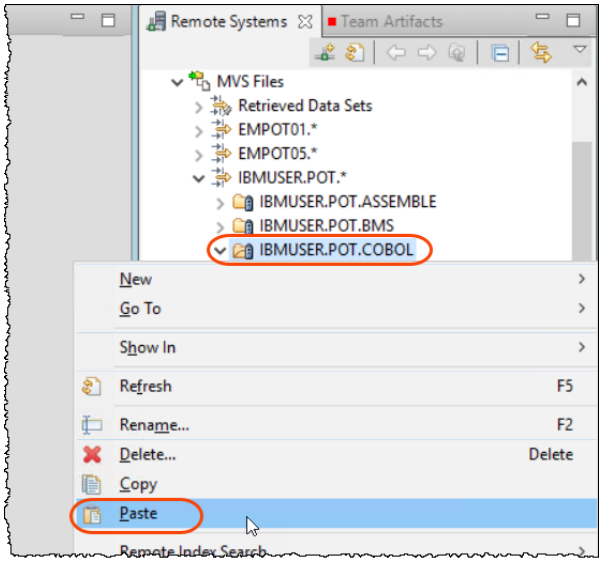
___ 6.2 Re-compile and link the changed program

You could use the IBM DBB (part of IDz) to do the compile and link as we already mentioned before, but on this exercise we decided to do it using the "old way, via JCL. Notice that the change was made at the local IDz project and the code must be moved to z/OS to compile it there. We had done similar task when we compiled/link the generated test case.

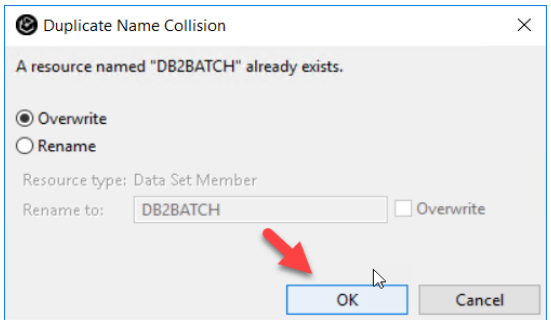
6.2.1 ⏭ Right click on **DB2BATCH.cbl** and select **Copy**



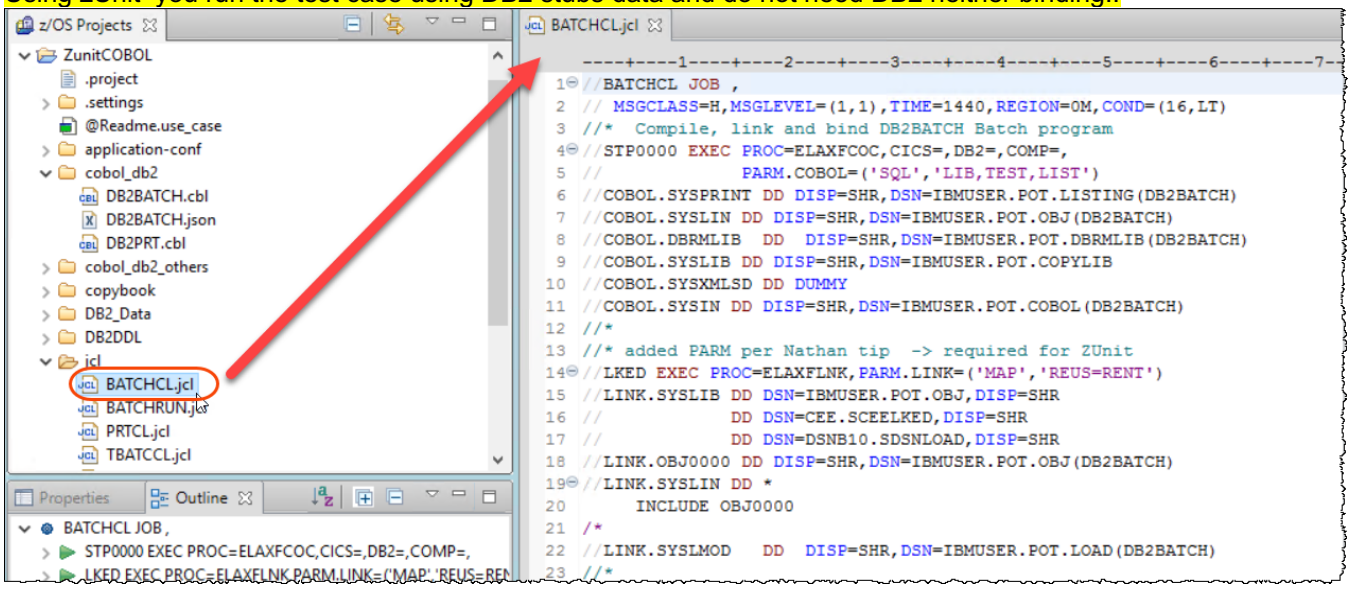
6.2.2 ⏭ Using the Remote systems (on right),
⏭ Right click on **IBMUSER.POT.COBOL** and select **Paste**



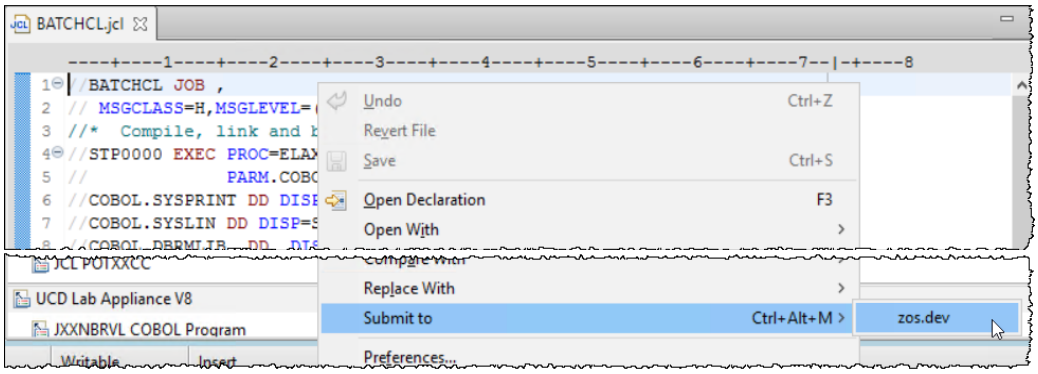
6.2.3 ⏭ Select **OK** to overwrite this member since it already existed on z/OS



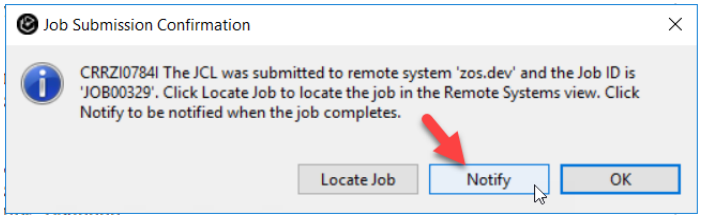
6.2.4 Under z/OS Projects expand **ZunitCOBOL/jcl**, and double click on **BATCHCL.jcl** .
This JCL will compile, link and bind the program being tested.
Notice that the DB2 bind would not be necessary unless you will really execute it.
Using zUnit you run the test case using DB2 stubs data and do not need DB2 neither binding..



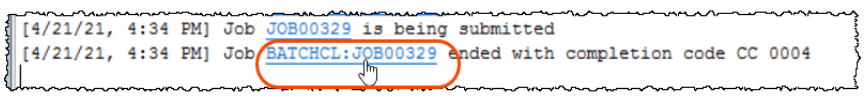
6.2.5 Right click on the JCL being edited and select **Submit to > zos.dev** for execution on z/OS



6.2.6 Click **Notify** to be notified when the execution is complete.

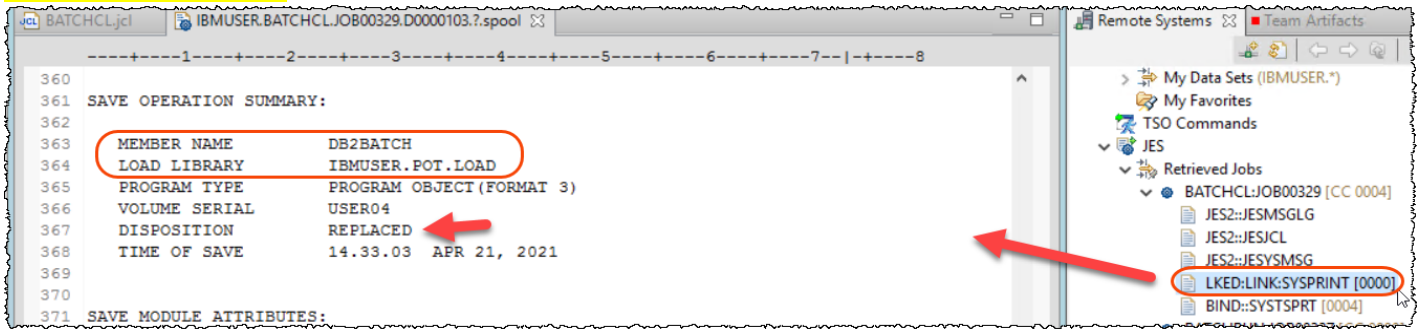


6.2.7 Under *Remote Console*, you will be notified when execution is completed.
The completion code must be 4.
Once the execution ends, **click** on the link **BATCHJCL:JOB00xxx**
(where 00x**xx** is a number) to point to the Job output. This number vary depending on z/OS.



6.2.8 Under *Remote Systems* view expand **BATCHJCL:JOB00xxx** and **double click LKED:LINK:SYSPRINT** step.

Scroll down the report displayed and you will see the load module **DB2BATCH** is replaced at **IBMUSER.POT.LOAD** .



6.2.9 Use **Ctrl + Shift + F4** to close all opened editors.

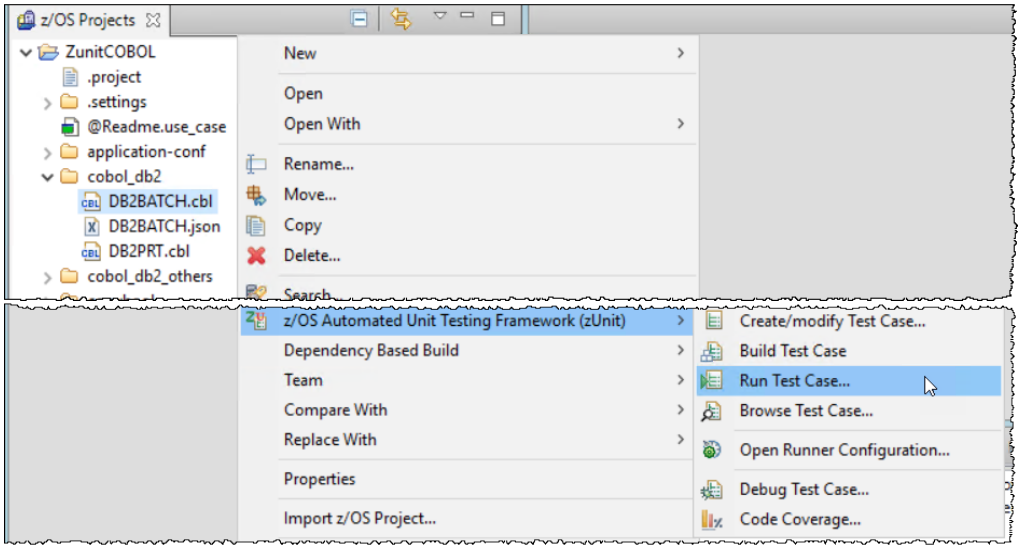
Section 7. Rerun the zUnit and verify that the bug is eliminated.

You will run the zUnit test case and verify that the program is fixed.

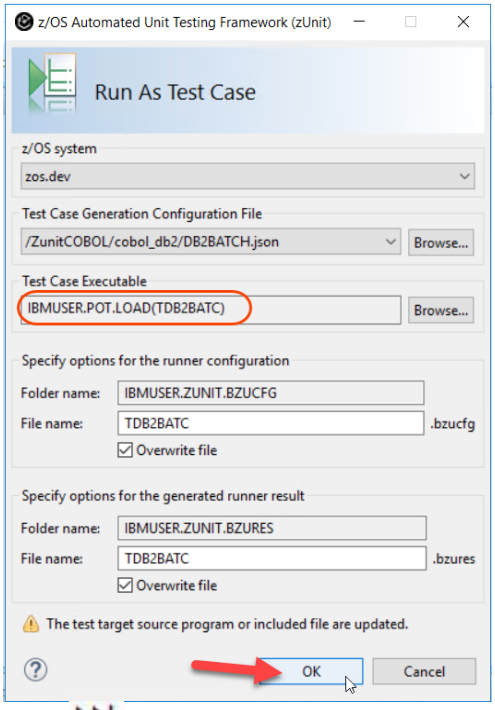
___ 7.1 Running the test case again

Since you fixed the bug the test case now should pass it.

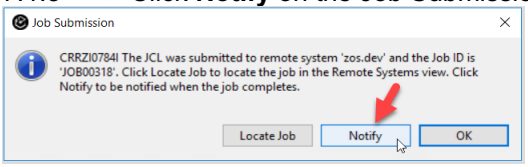
7.1.1 On the z/OS Projects view, select **DB2BATCH.cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)->Run Test Case..**



7.1.2 Verify that the test case load module **IBMUSER.POT.LOAD(TDB2BATC)** is selected and click **OK**

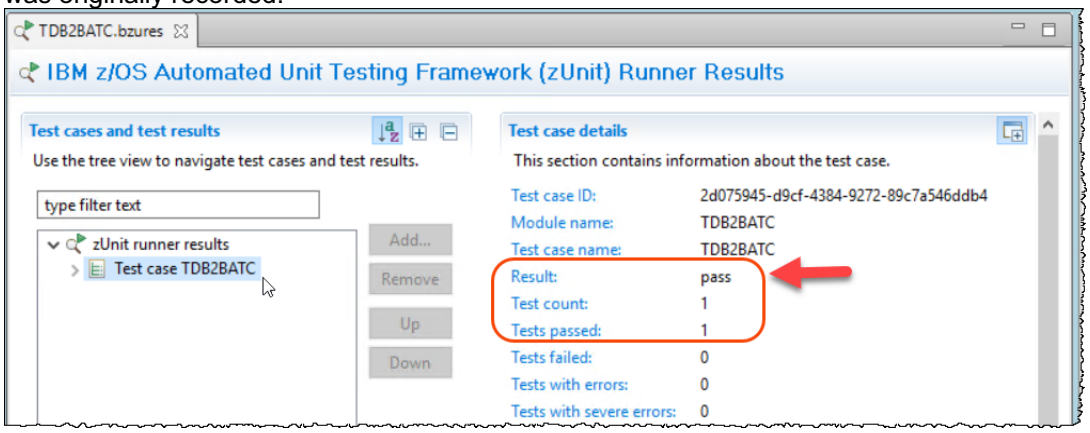


7.1.3 Click **Notify** on the Job Submission Confirmation dialog.



7.1.4 When the test run completes, the test results will be displayed, and it showed that one test ran and it failed. Also notice that the completion code for this JCL execution is now 0000 since the test case passed

Click on **Test case TDB2BATC.** The test case now passed since the bug is fixed and results match what was originally recorded.



Congratulations! You have completed the Lab 2B.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

Please Recycle