.LAB 6C – (OPTIONAL) Using IBM zUnit to Unit Test a COBOL CICS/DB2 application using Local project (60 minutes)

Updated February 23, 2021 by Regi –

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.

The main difference between this lab and the LAB 6B is that here we will use local projects on IDz and IBM DBB (Dependency Based Build) instead of JCL ro compile the COBOL programs.

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. **Record interaction with the application.**
   → You will record an interaction with the COBOL CICS/DB2 program.
3. **Generate, build and run the unit test**
   → You will compile and link-edit the generated unit test program, followed by running the unit test.
4. **Introduce a bug in 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.
5. **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.

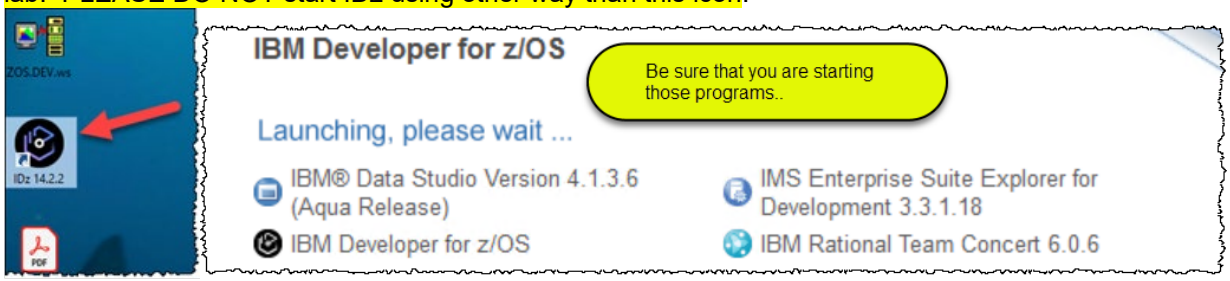
___ 1.1 Connect to z/OS and emulate a CICS 3270 terminal

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

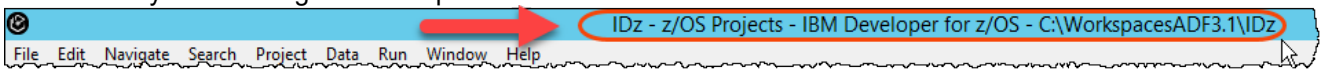
1.1.1 ⏭ Start *IBM Developer for z Systems version 14* if it is not already started

⏭ Using the desktop double click on **IDz 14.2.2** icon.

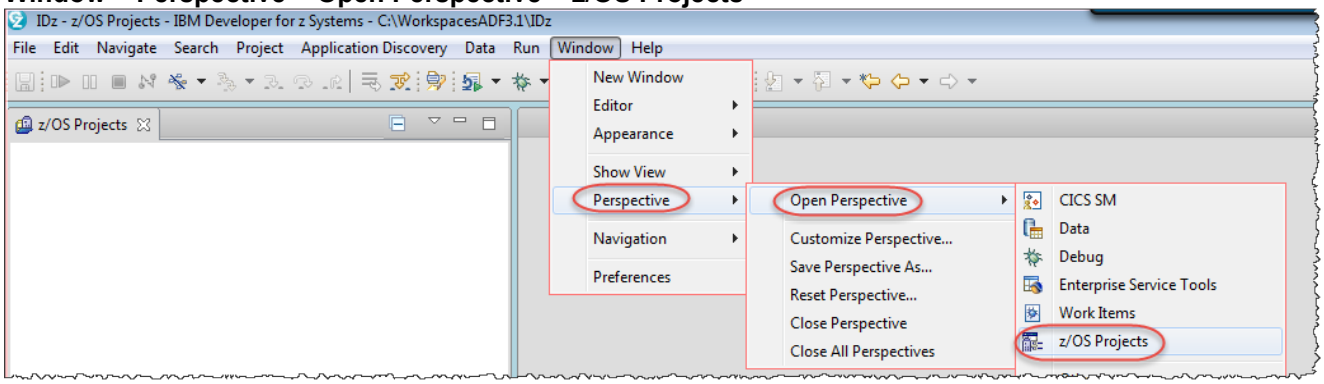
IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon.



⏭ Be sure you are using this workspace:

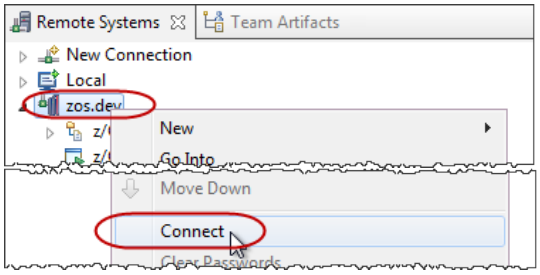


1.1.2 ⏭ Open the **z/OS Projects** perspective by selecting
Window > Perspective > Open Perspective > z/OS Projects



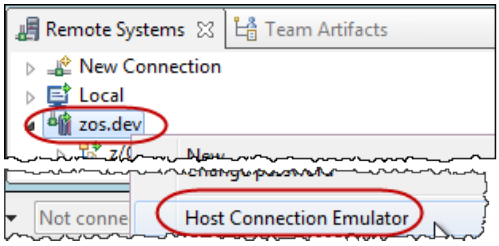
1.1.3 On this lab you will use userid **ibmuser** .
If you are connected as ibmuser, jump to step 1.1.5 Otherwise**.**

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

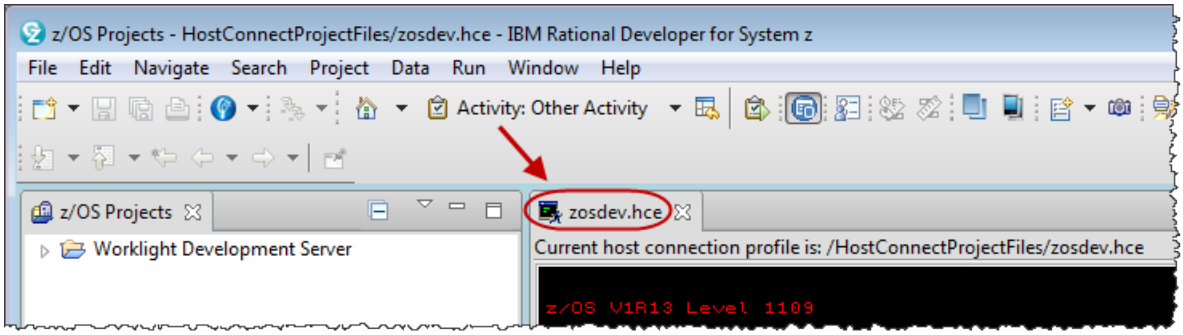


1.1.4 ⏭ Type **ibmuser** as userid and **sys1** as password.
The userid and password can be any case; don't worry about having it in UPPER case.
Click **OK** to connect to z/OS.

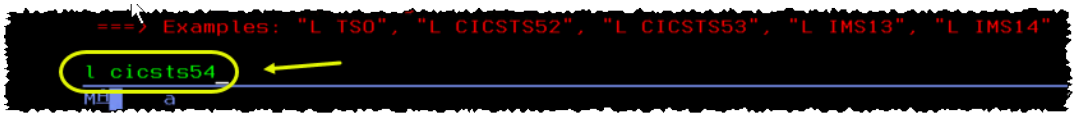
1.1.5 ⏭ Using the *Remote Systems* view, right click on **zos.dev** and select **Host Connection Emulator.**



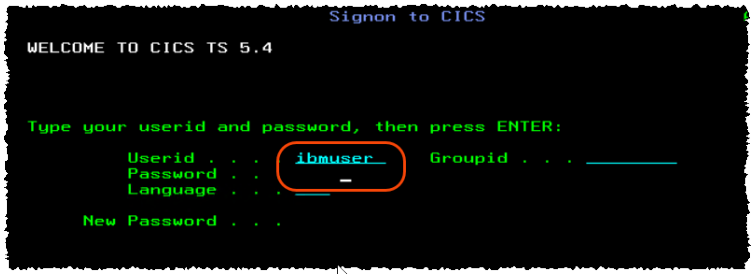
1.1.6 ⏭ Since you will need more space, **double-click** on the ***zos.dev.hce*** title



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 **`ibmuser`** and password **`sys1`** and press **Enter**.

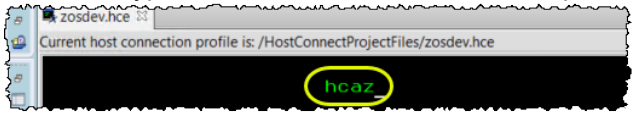


1.1.9 The sign-on message is displayed

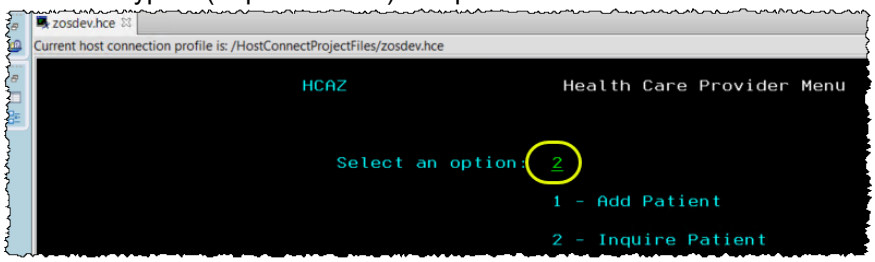
___ 1.2 Run CICS transaction HCAZ

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.

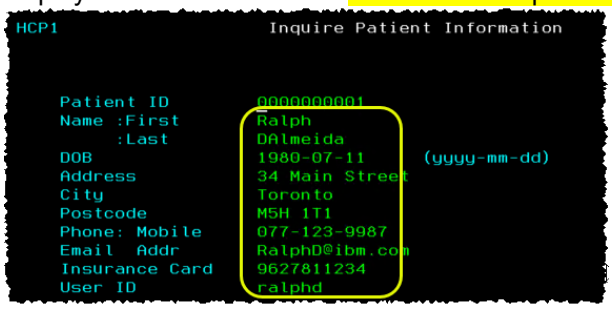
1.2.1 ⏭ Type the CICS transaction **hcaz** and press the **Enter** key.



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

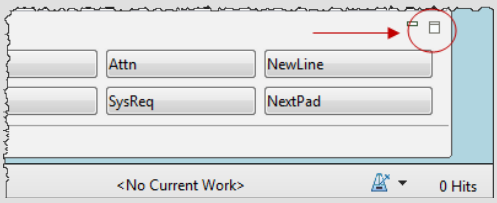


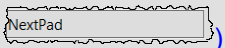
1.2.3 ⏭ Type **1** for Patient ID and press Enter. The program will read the patient from a DB2 table and display the customer details. Remember the patient first name: **Ralph**



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 → zosdev.hce. 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 – Record data interaction using the CICS application.

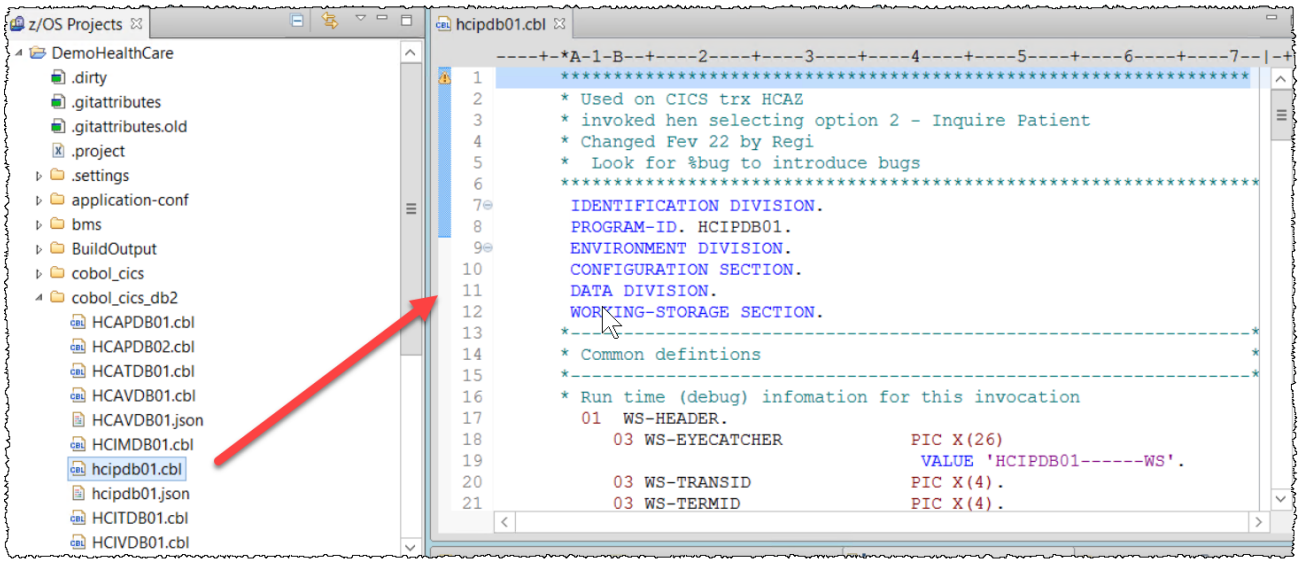
Using IDz you will record the interaction with the COBOL/DB2 program that reads a patient from a DB2 table.

___ 2.1 Understanding the COBOL program that reads from DB2 table

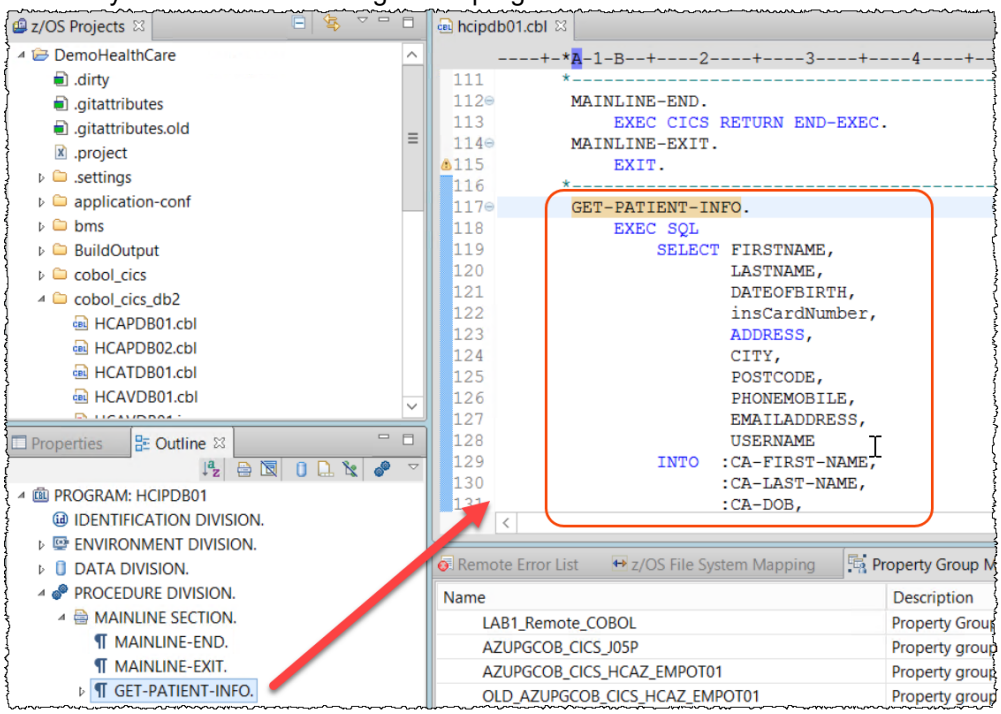
The COBOL code that reads the patient from the DB2 tables is the program ***HCIPDB01***

2.1.1 Using *z/OS Projects* view
double click on **hcipdb01cbl** under **DemoHealthCare/cobol_cics_db2.**

This is the program that you will update later on.

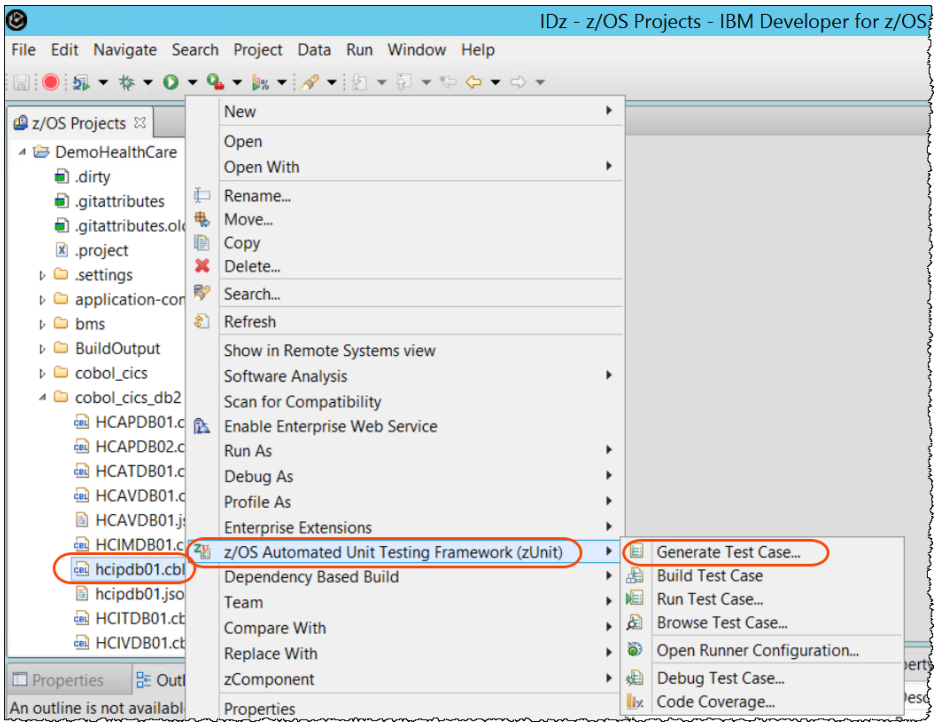


2.1.2 Using the Outline view on left expand **PROCEDURE DIVISION** and click on **GET-PATIENT-INFO.**
This is where the patient data is read from the DB2 table.
Later on you will introduce a bug in this program..



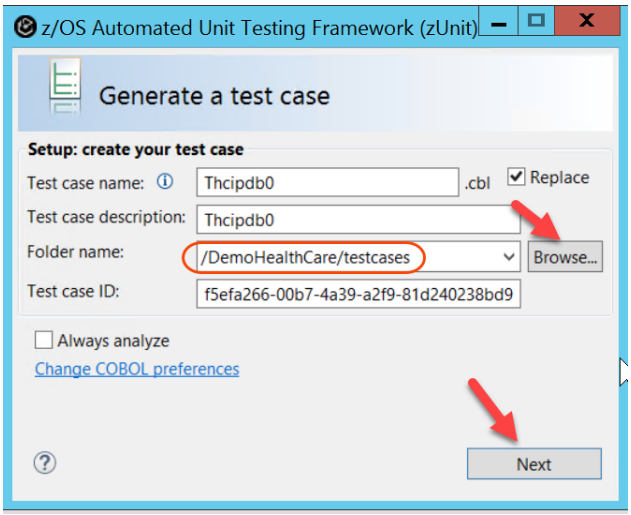
___ 2.2 Recording the COBOL program that sends the message

2.2.1 To start the recording, right click on **hcipdb01.cbl**
and select **z/OS Automated Unit Testing Framework (zUnit)->Generate Test Case..**



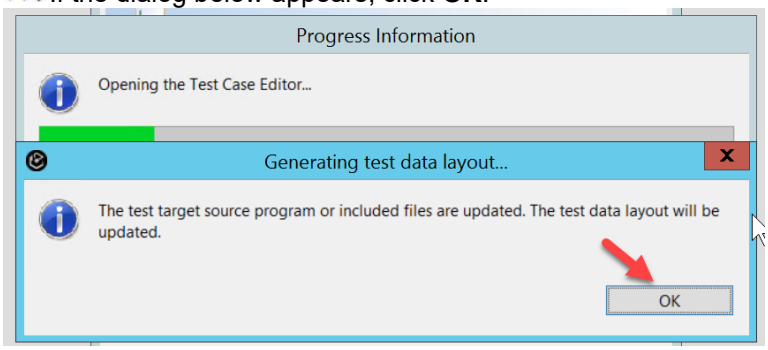
2.2.2 This opens a dialog where you can name your test case.

⏭ Using the Browse button select **/DemoHeathCare/testcases** as folder name, ignore Test case ID and click **Next**.

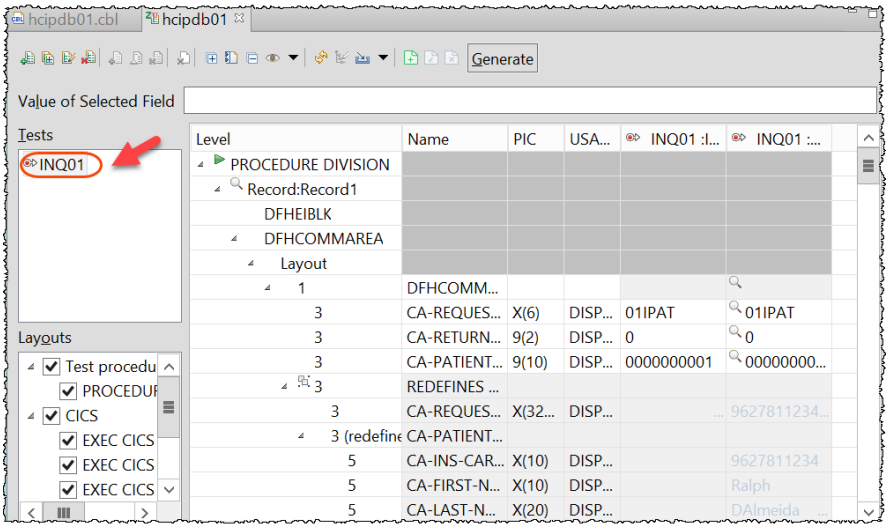


2.2.3 This operation will generate the test data layout on the local workspace.

⏭ If the dialog below appears, click **OK**.

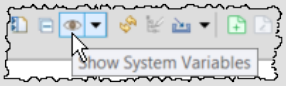


2.2.4 This will open the **Test Case Editor**, as shown below. INQ01 may or may not be on your screen. If **INQ01** is there you will delete on next step.



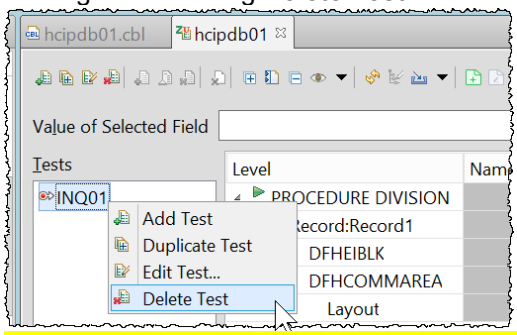
Understanding the test case editor

- 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

2.2.5 Since we will be recording the test data, delete the **INQ01** or any other entry (if it exists) by right clicking and selecting **Delete Test**

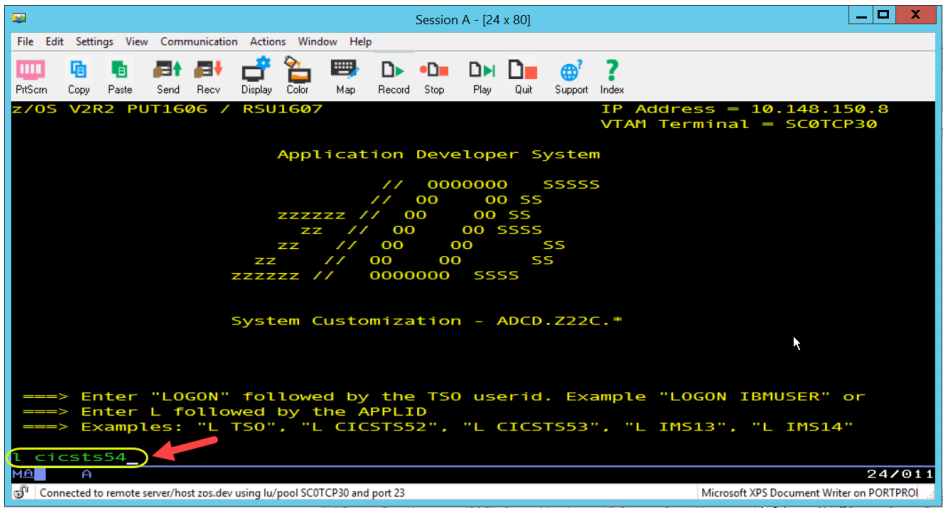


Notice that on section 1 we used the IDz emulator and now you will use the IBM PCOM emulator. Any 3270 emulator can be used.. We just show here a different way to emulate a 3270 terminal.

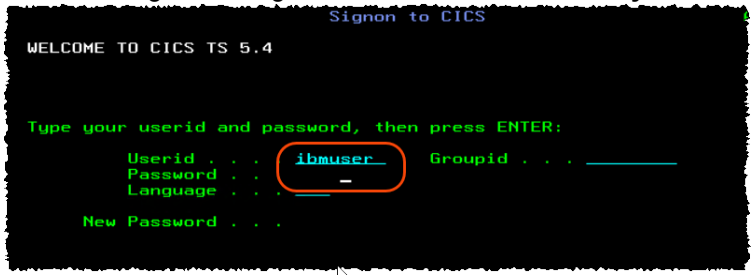
2.2.6 Bring up a 3270 terminal emulator clicking on the **host emulator icon** on the Windows task bar.

This opens the host emulator.

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



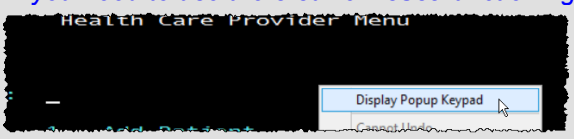
2.2.8 ⏭ Sign on using **ibmuser** as the userid and **sys1** as the password.



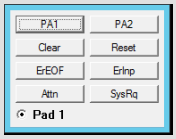
2.2.9 ⏭ Once you see the sign-on is complete message, enter **hcaz** and press the right **Ctrl** key.

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*



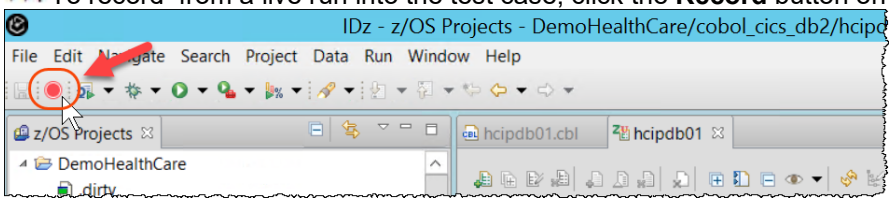
and



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

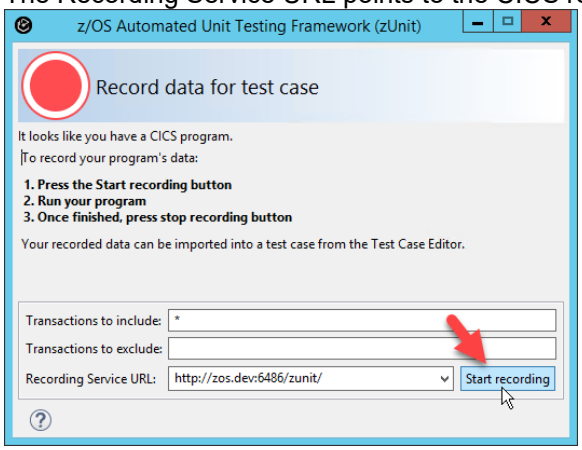
2.2.10 ⏭ Minimize the terminal emulator and go back to IDz dialog.

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

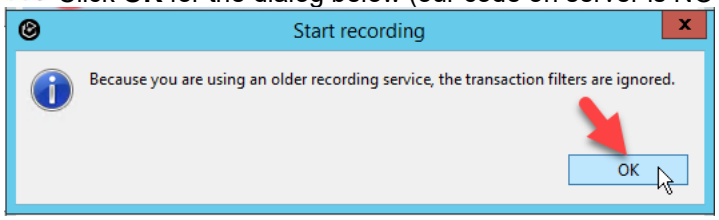


2.2.11 ⏭ In the dialog that comes up, click on **Start recording**.

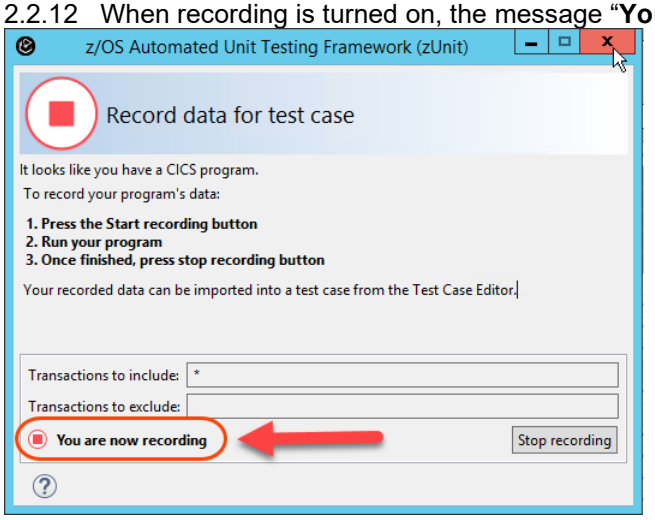
The Recording Service URL points to the CICS region where the live run is recorded.



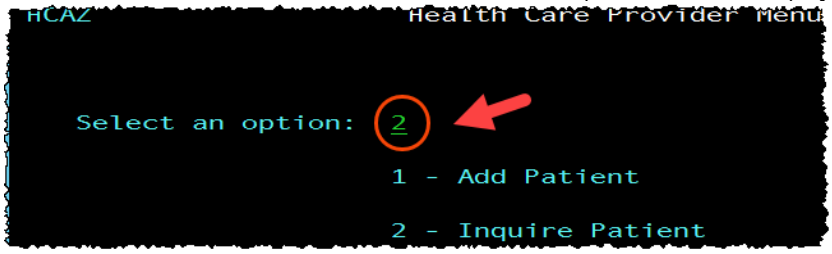
⏭ Click **OK** for the dialog below (our code on server is NOT the latest level).



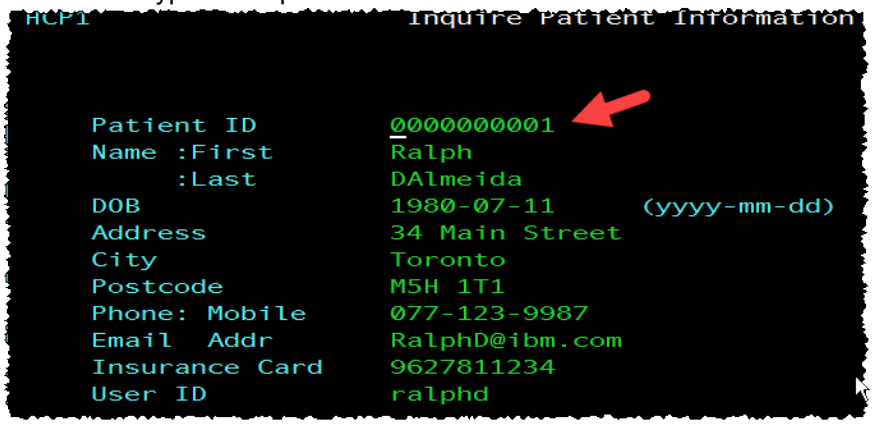
2.2.12 When recording is turned on, the message "**You are now recording**" appears.



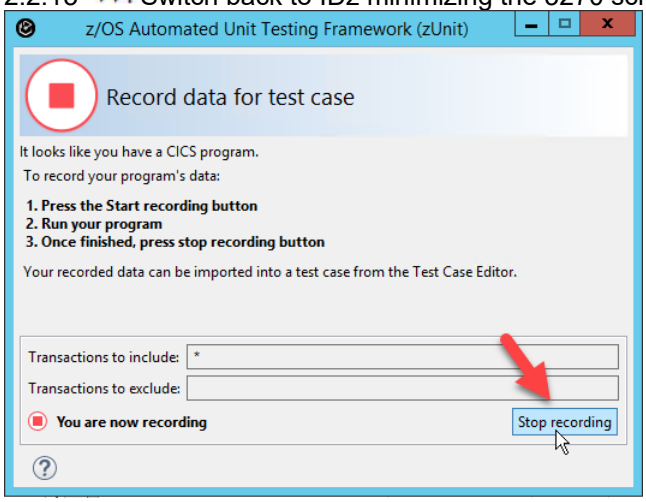
2.2.13 ⏭ Switch to the 3270 terminal emulator (that is minimized), type **2** (Inquire Patient) and press **enter**.



2.2.14 ⏭ Type **1** and press **enter**

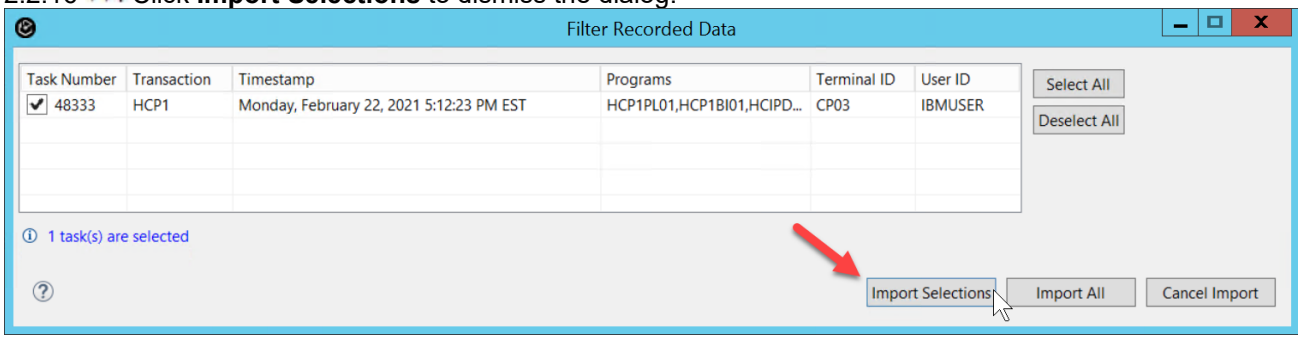


2.2.15 ⏭ Switch back to IDz minimizing the 3270 screen and click **Stop recording**.



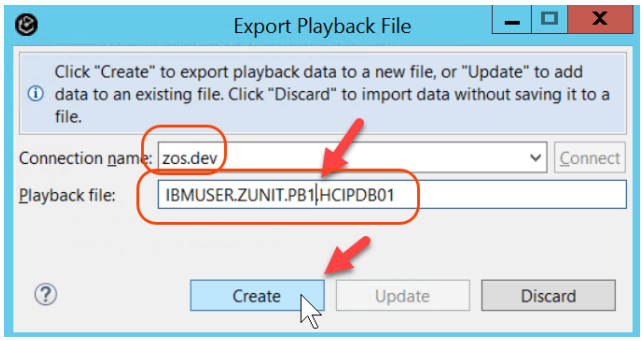
The dialog Filter Recorded Data is displayed.

2.2.16 ⏭ Click **Import Selections** to dismiss the dialog.

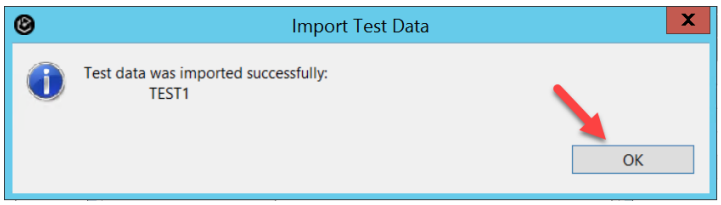


2.2.17 ⏭ Using drop down select the **zos.dev** as Connection name
Modify the Playback file name to **`IBMUSER.ZUNIT.PB1.HCIPDB01`**
Click **Create** to export playback data to a z/OS data set

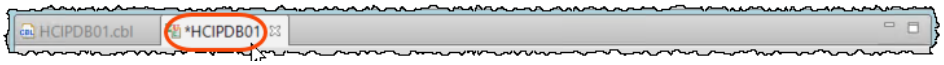
.



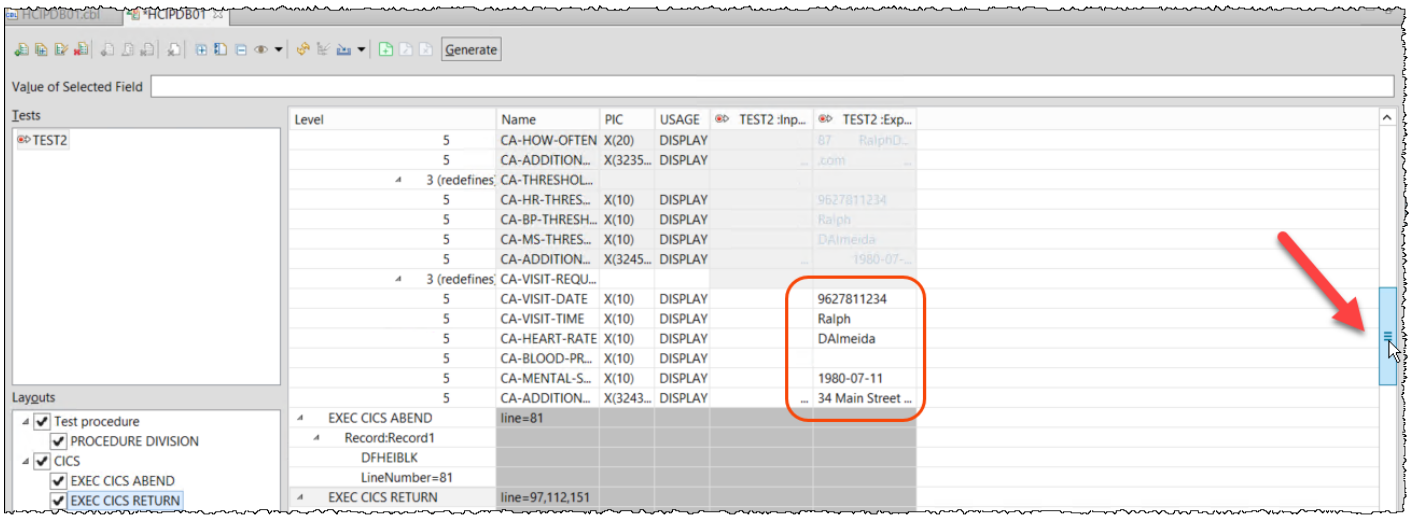
2.2.18 ⏭ Click **OK** to dismiss the dialog. You may have **TEST2** instead of TEST1.



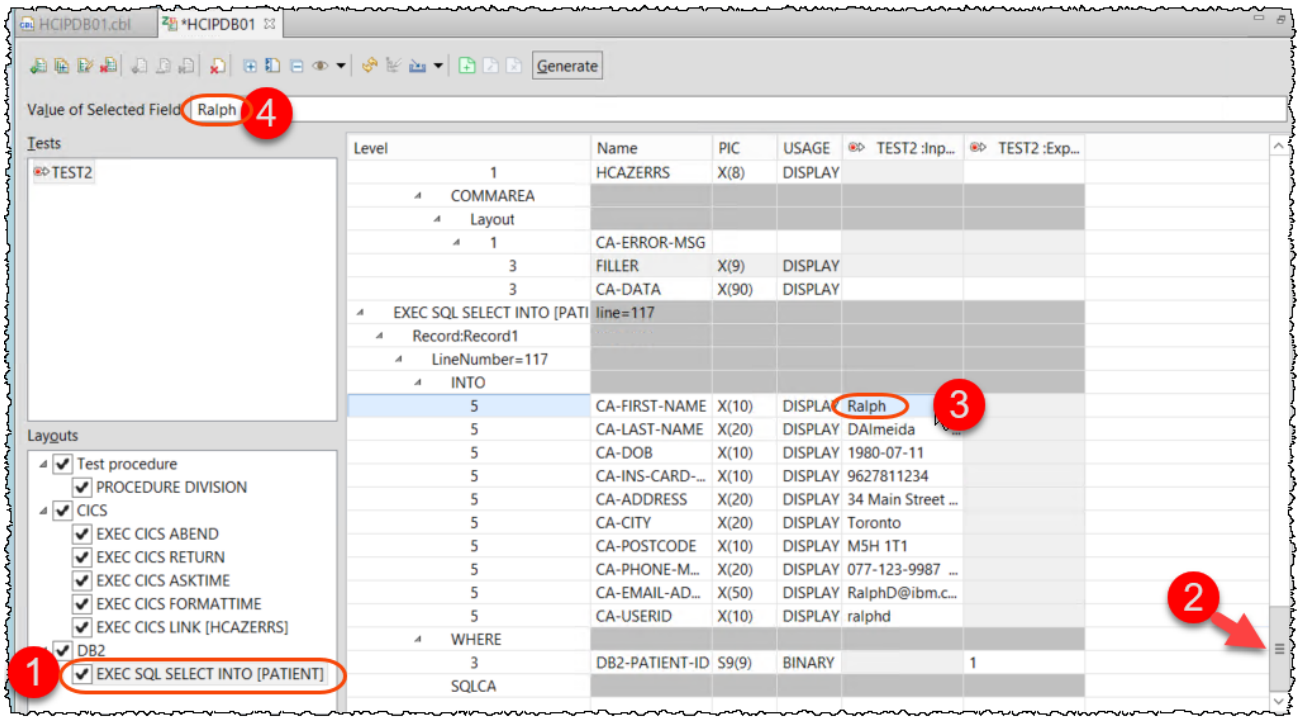
2.2.19 ⏭ Double click on the **HCIPDB01** title to enlarge the view.



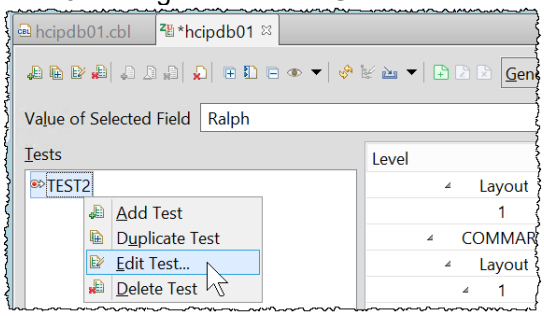
2.2.20 You now see a new test created in the editor and populated with the values from the live run.
⏭ **Scroll down** the editor and notice the data displayed on the screen that was captured..



2.2.21 ⏭ 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”

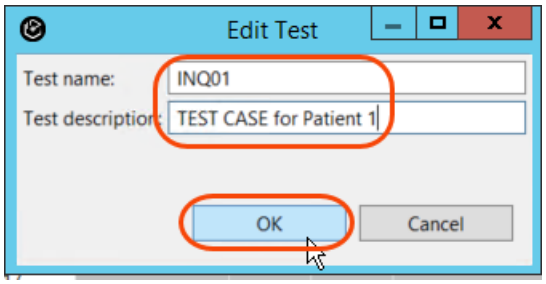


2.2.23 ⏭ Right click on **TEST2** and select **Edit Test**



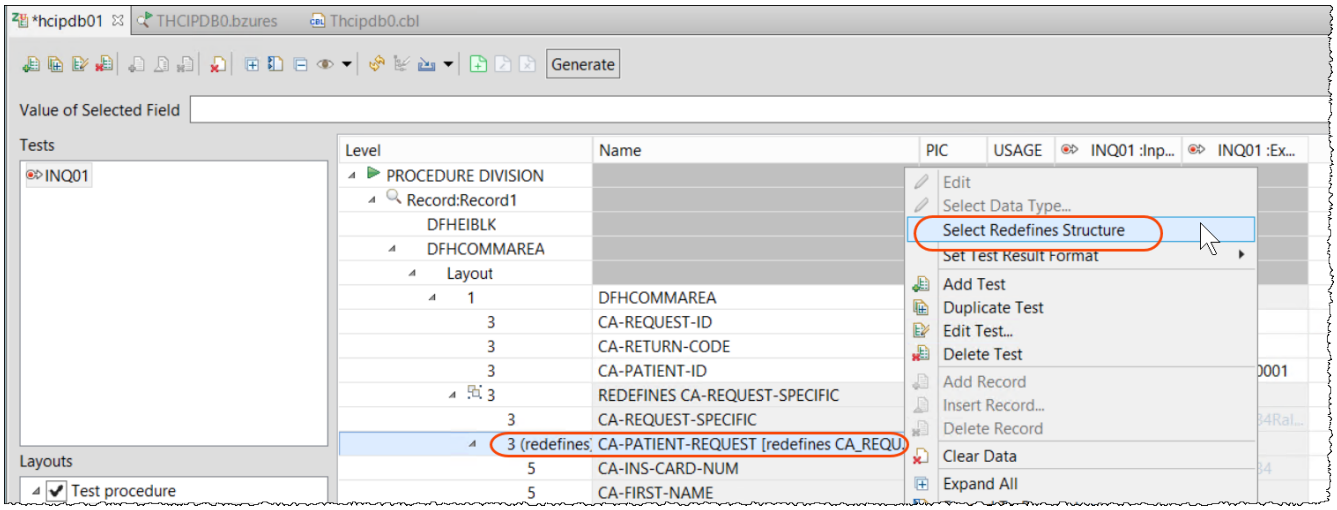
2.2.24 It is a good practice give a name for the test case.

⏭ Use a name like **INQ01** and a description like "**TEST CASE for Patient 1**": . Click **OK**



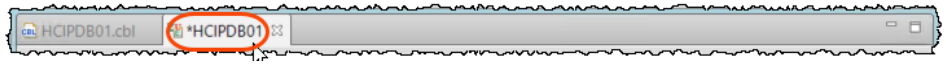
2.2.25 You may notice that this program has many redefines. You may need to identify which is the area being used on this program execution. In our example is the "inquiry patient" area being redefined.

⏭ **Scroll up** until you find **"(redefines) CA-PATIENT-REQUEST**" (the first REDEFINES).
Right click on it and click **Select Redefines Structure**



2.2.26 ⏭ Press **Ctrl+S** to save any changes.

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

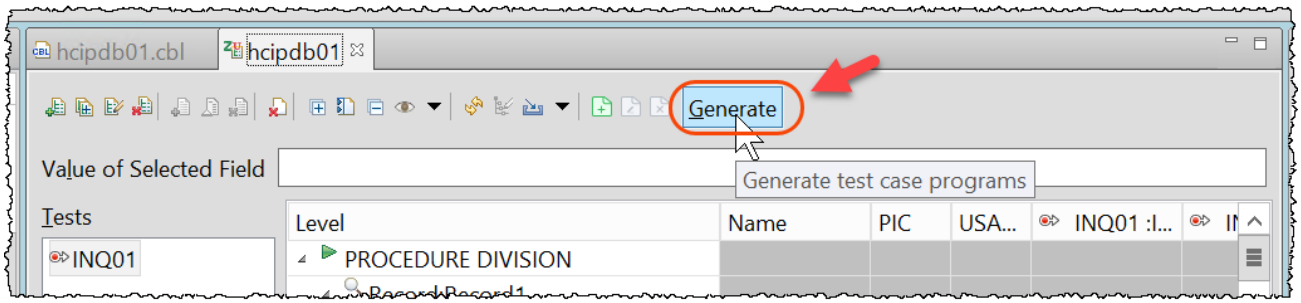


Section 3. Generate, build and run the unit test.

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

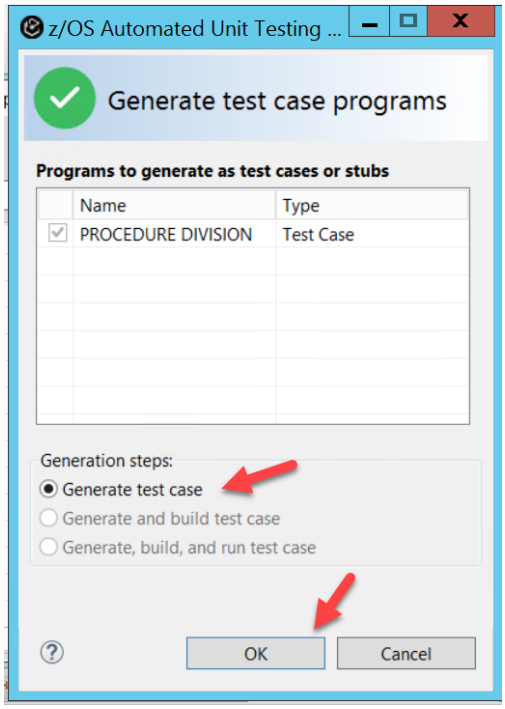
___ 3.1 Generating the test case program.

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 program.



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

3.1.2 ⏭ Select **Generate test case** and then click **OK**.

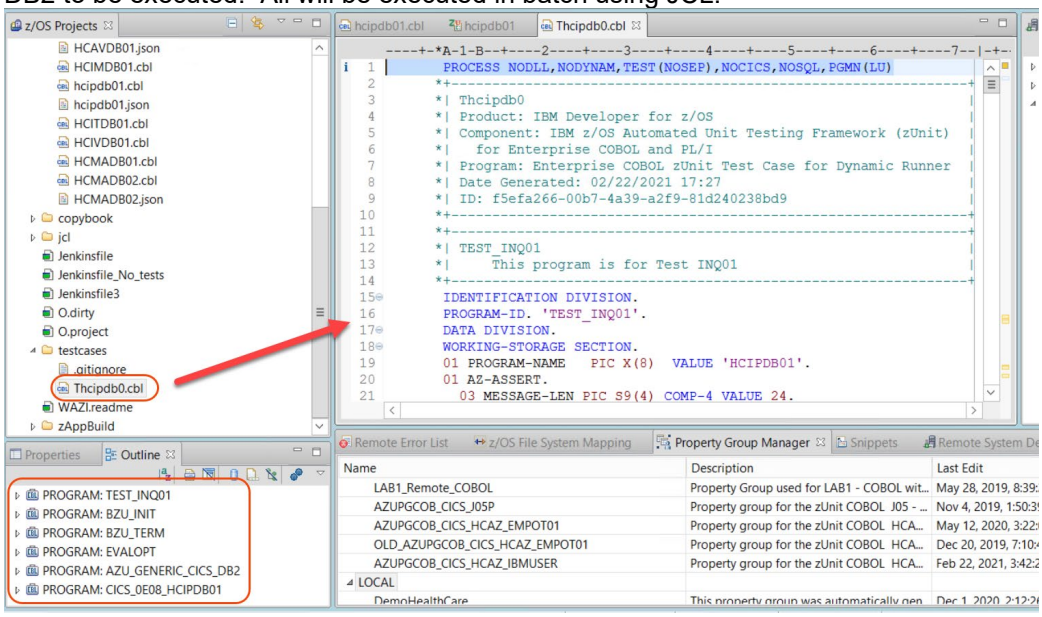


3.1.3 The COBOL program that will run the test case is generated under the folder *testcases*.

⏭ Expand **testcases** and double click on **Thcipdb0.cbl** to verify the generated program.

Notice that in fact 6 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.



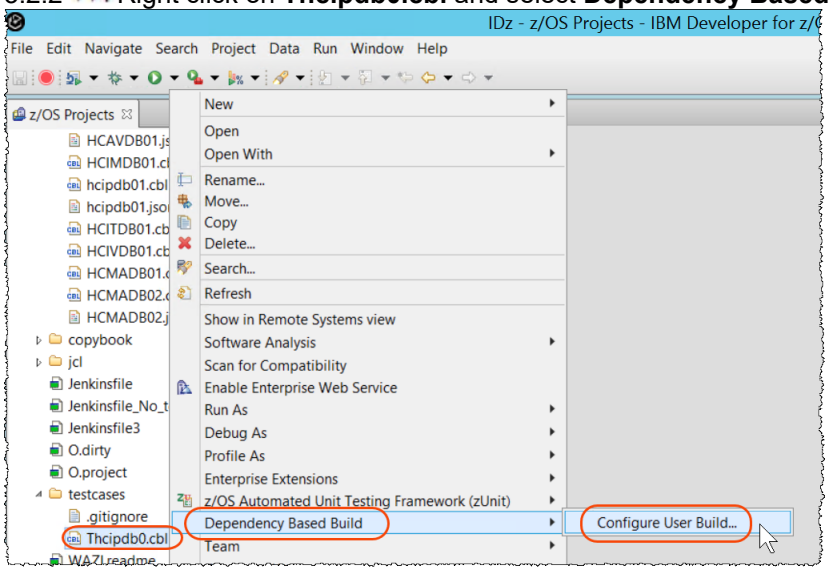
___ 3.2 Building the generated test case programs using IBM DBB .

The 6 generated COBOL programs need to be compiled/link edited to be executed. This must be done on z/OS using the COBOL compiler.

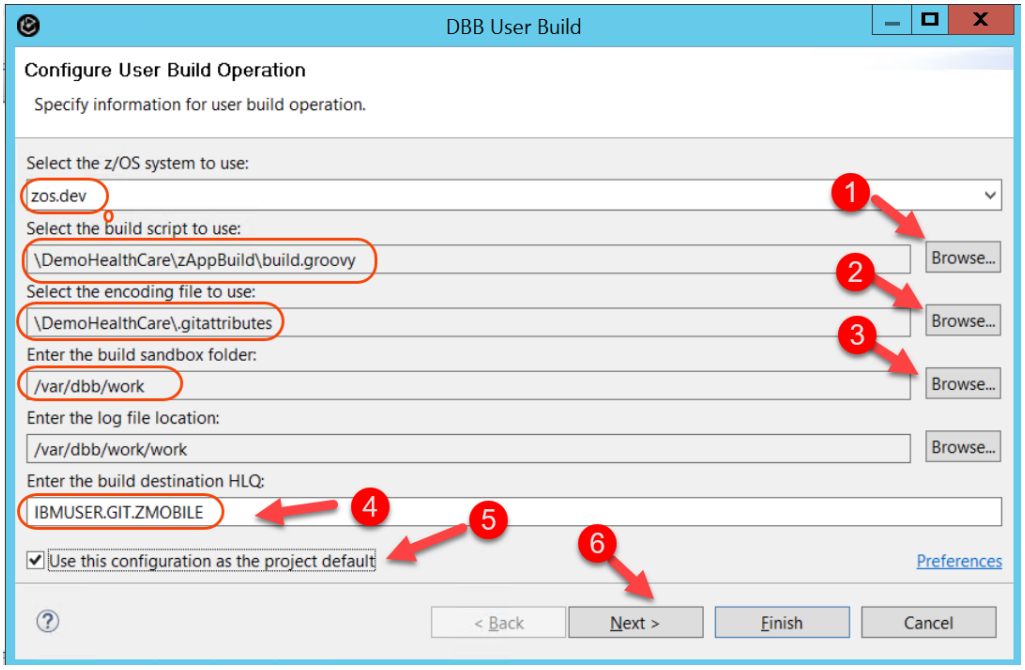
To make this easier we will use the IBM DBB. We provided the required framework to make this possible. But note that this could be done using any other way, including via JCL normal compilation.

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

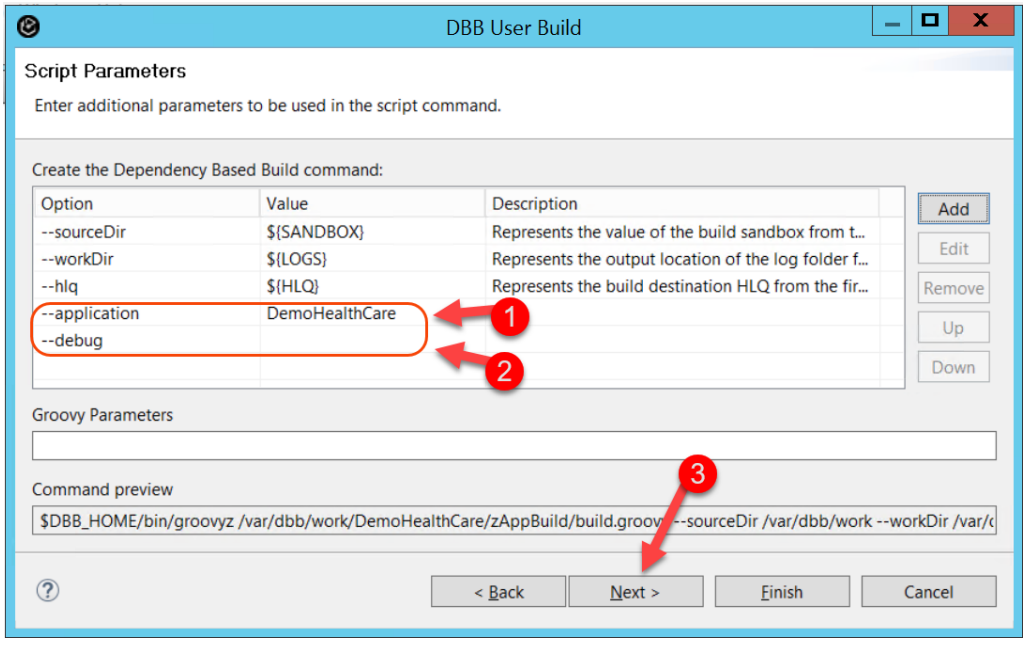
3.2.2 ⏭ Right click on **Thcipdb0.cbl** and select **Dependency Based Build > Configure User Build.**



3.2.3 ⏭ Use the Browse buttons to specify the values as below.
Type the build destination HLQ as **IBMUSER.GIT.ZMOBILE**, select **Use this configuration as the project default** and click **Next**

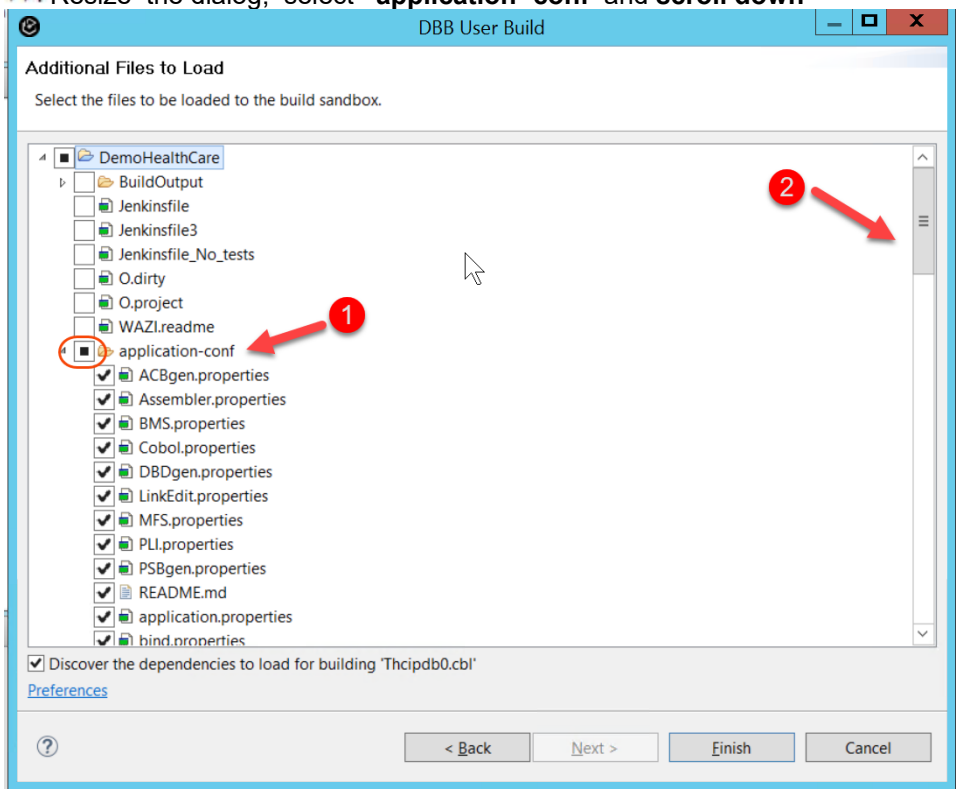


3.2.4 ⏭ Click **Add** to insert the parameters as below.
Notice that the application name **DemoHealthCare** is mixed case
Click **Next**

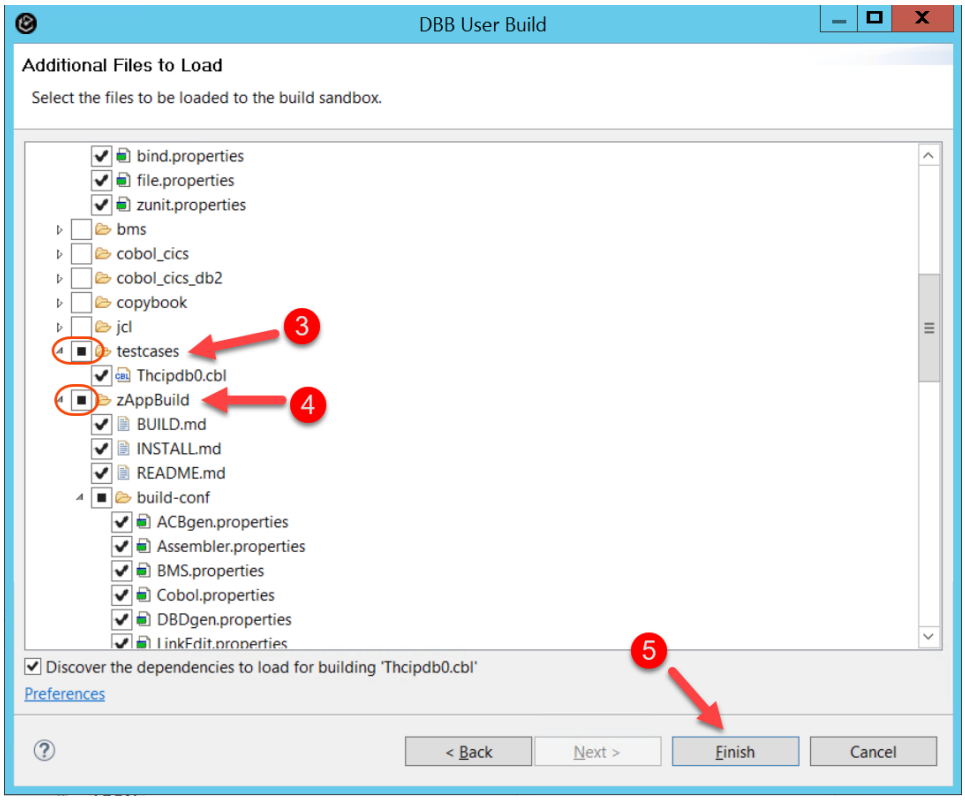


3.2.5 You need to move the assets to z/OS UNIX Files

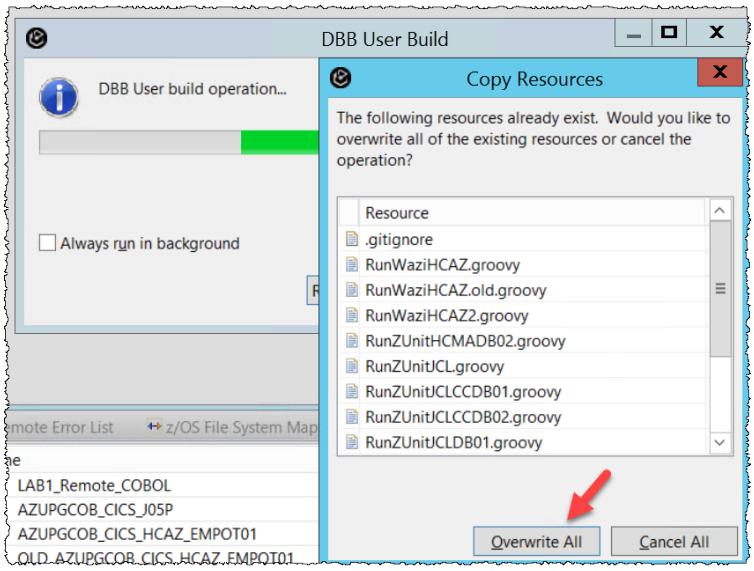
⏭ Resize the dialog, select **application -conf** and **scroll down**



3.2.6 ⏭ Be sure that **testcases** and **zAppBuild** boxes are selected and click **Finish**

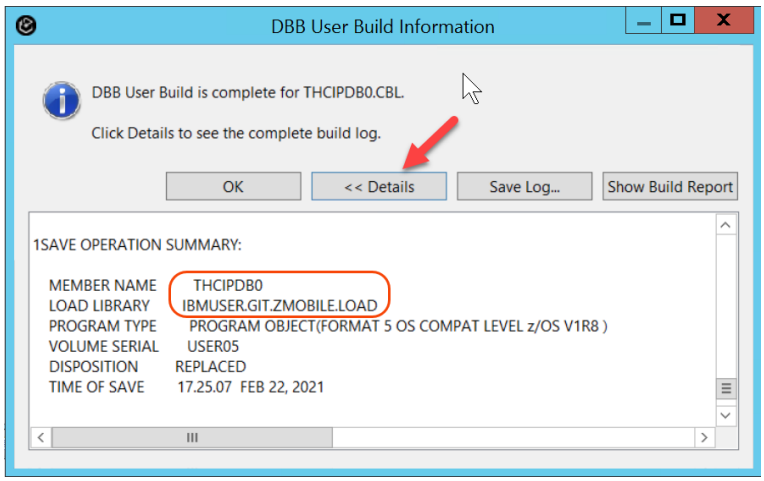


3.2.7 ⏭ Select **Overwrite All** if for the various dialogs. It's possible that those assets were already moved to the z/OS UNIX files. See one example below.

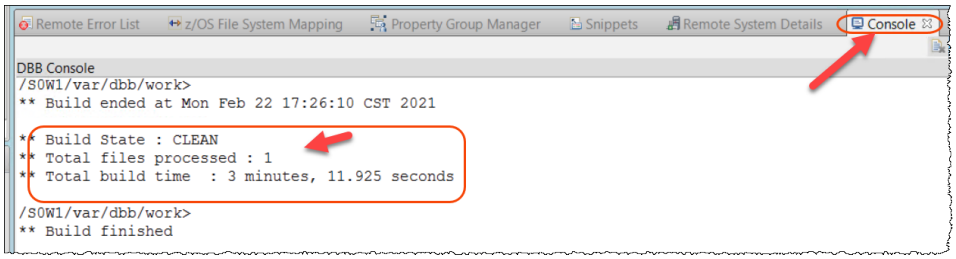


3.2.8 This operation will compile and link the 6 generated COBOL programs and it may take up to 4 minutes. When it starts you can see the ***Console view*** opening in the bottom.

⏭ When finished the dialog below will be displayed and you can click on **Details** and **OK** after you verify that the load module was created.



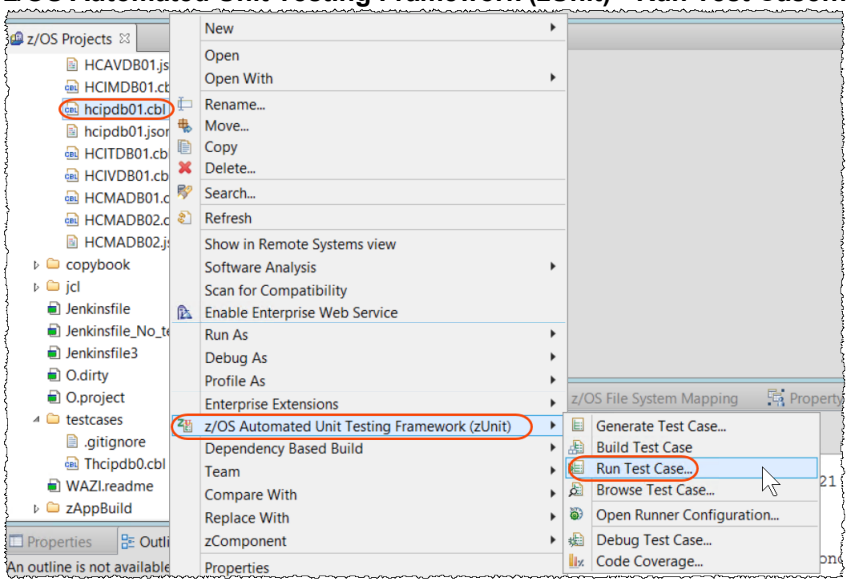
The ***Console view*** also show the results as below:



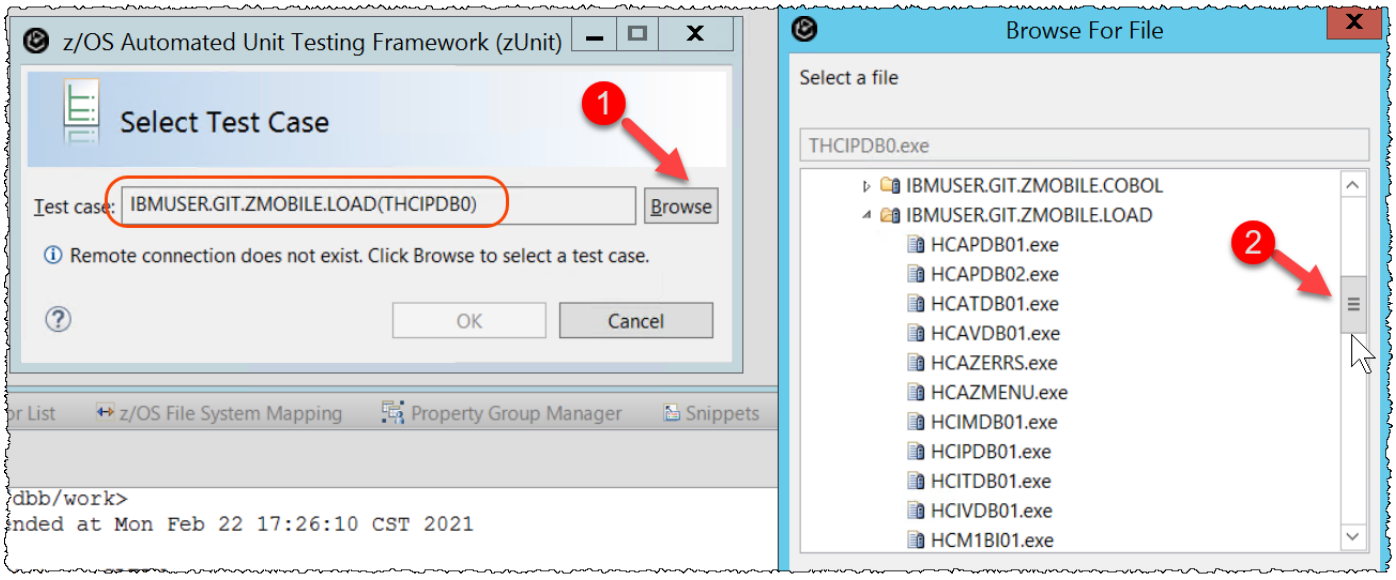
___ 3.3 Running the test case,

Once you have the test cases load module created you can run the test case against the 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 Right click on **hcipdb01.cbl** and select
z/OS Automated Unit Testing Framework (zUnit)->Run Test Case…

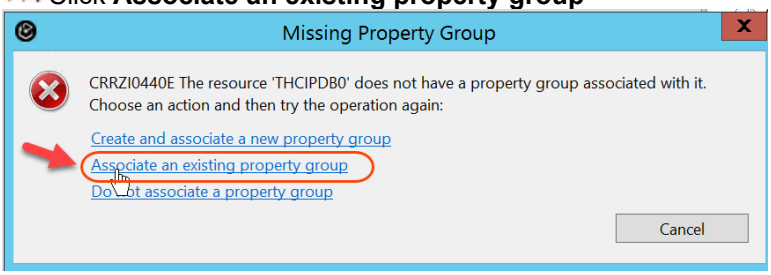


3.3.2 Use the **Browse** button to select the PDS and load module where the test case was created. You will find the PDS **`IBMUSER.GIT.ZMOBILE(THCIPDB0)`** under **My Data Sets** .. Click on **`THCIPDB0`** to select it.

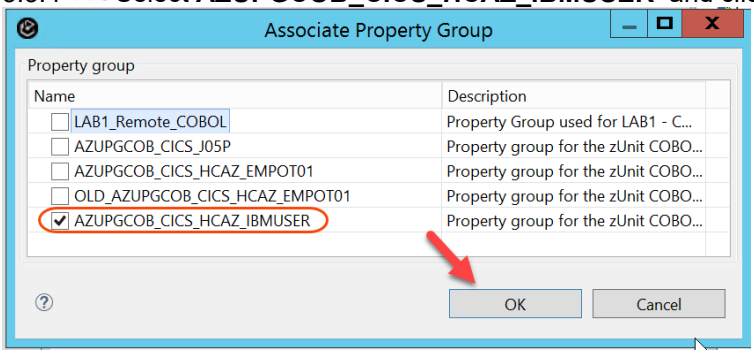


3.3.3 The *Missing Property Group* dialog will prompt.

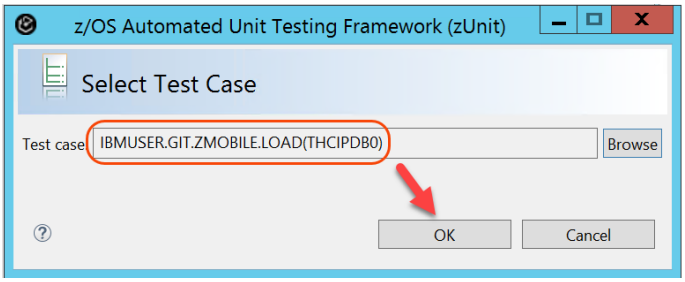
⏭ Click **Associate an existing property group**



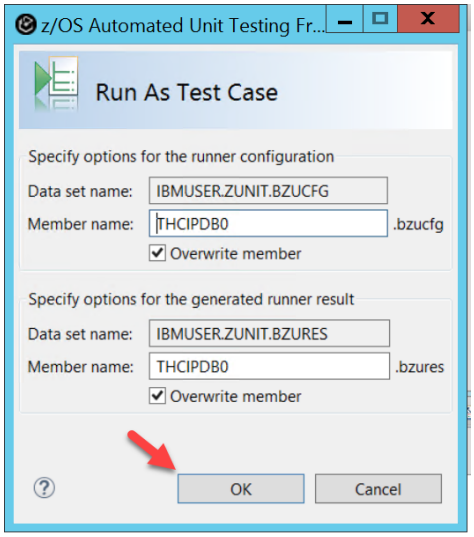
3.3.4 ⏭ Select **AZUPGCOB_CICS_HCAZ_IBMUSER** and click **OK**



3.3.5 ⏭ Click **OK**

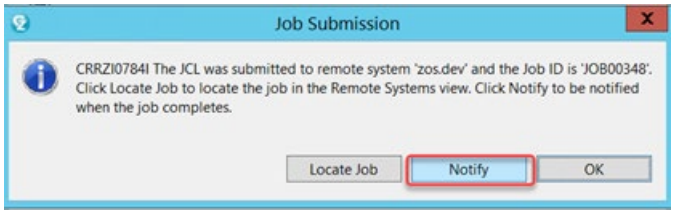


3.3.6 ⏭ Click **OK** to generate and submit a JCL for batch execution.

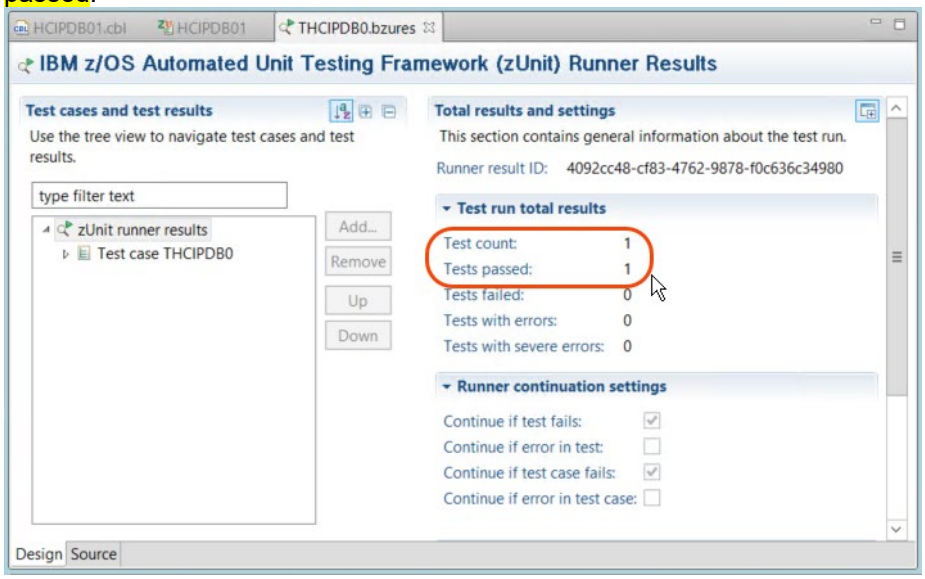


A **Progress information** dialog on the building will open.

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



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



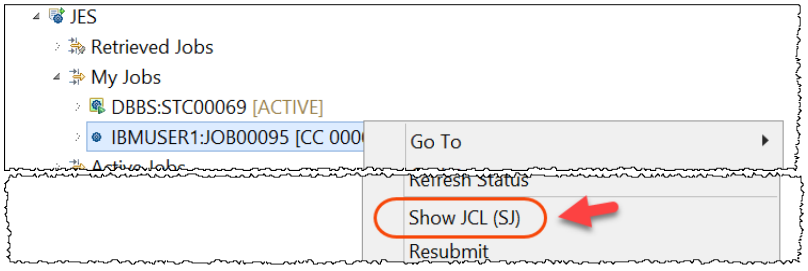
You have now created a test case with data imported from a recorded run and have been successful in testing a CICS program without the need of a CICS and DB2 environment.

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

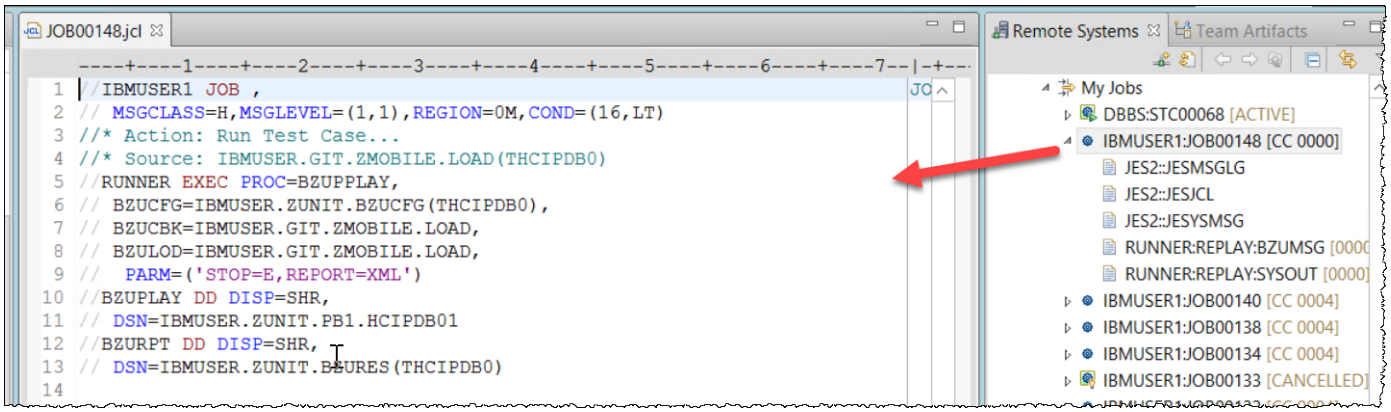
___ 3.4 Verify the JCL submitted

To see the JCL submitted on the previous execution

3.4.1 Using *Remote Systems* view on right, under **JES** expand **My Jobs**, right click on the last executed and select **Show JCL (SJ)**



3.4.2 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 CICS or DB2 environments in that execution.



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

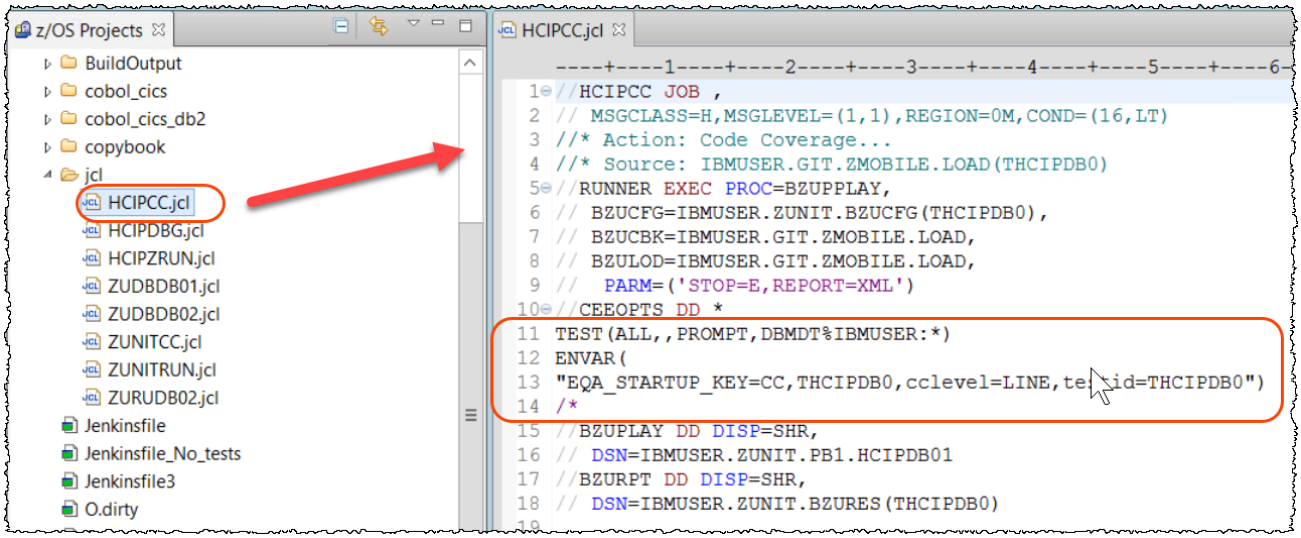
___ 3.5 Running zUnit test case Code Coverage

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.

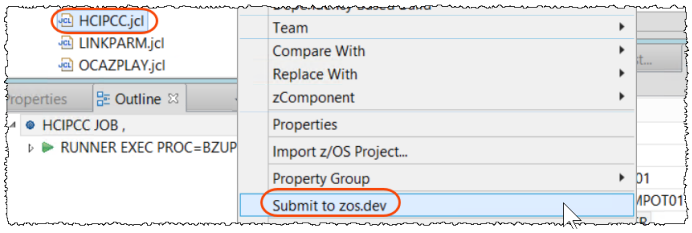
Notice that you are doing the code coverage using a batch job without need to have CICS and DB2 active, what is very handy..

3.5.1 We saved the JCL showed on step 3.4.2 and added the information to run the code coverage. This JCL can be found at **`jcl/HCIPCC.jcl`**

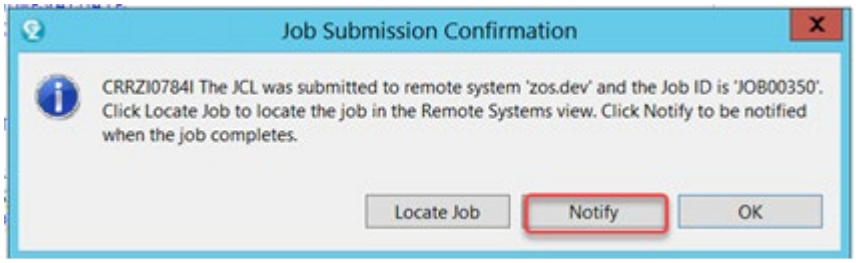
Double click on **`jcl/HCIPCC.jcl`** and verify this JCL.



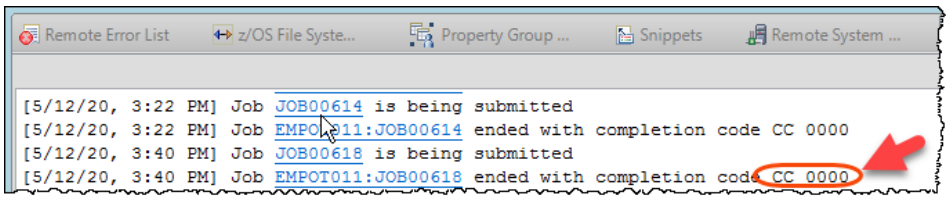
3.5.2 ⏭ Right click on **`HCIPCC.jcl`** and select Submit to zos.dev



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

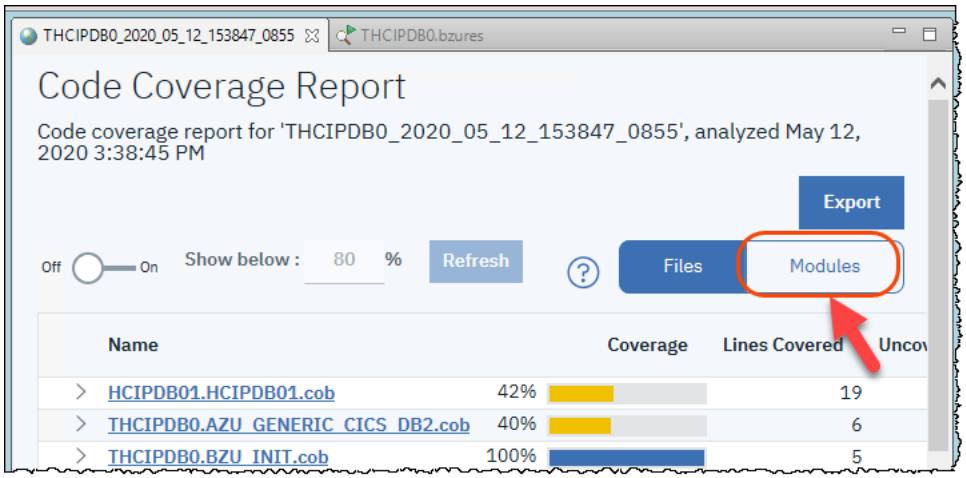


⏭ Make sure the job completes with completion code of **0000**.

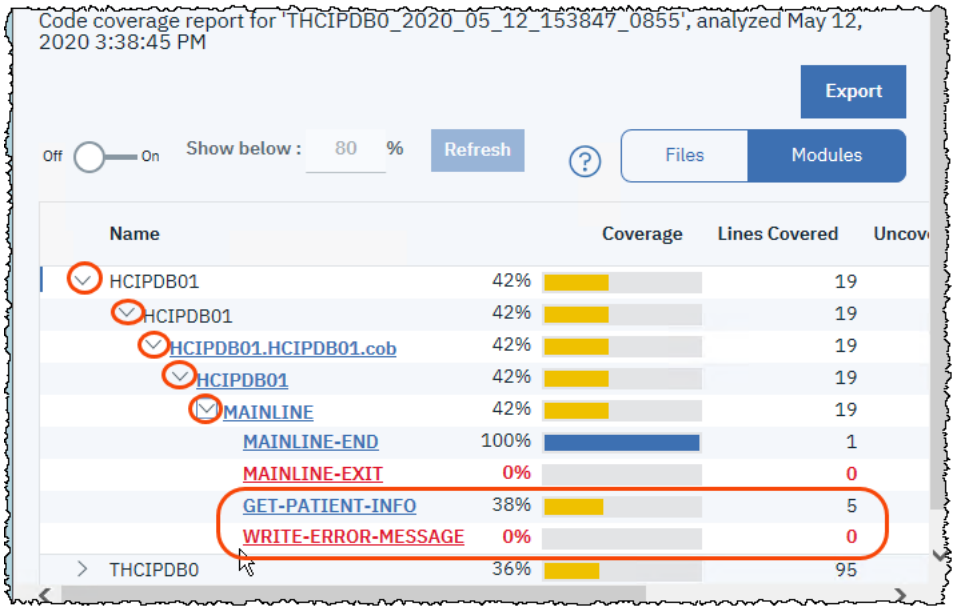


3.5.5 The code coverage report shows all COBOL programs executed for this specific test case.
It shows the coverage of the COBOL programs generated for zUnit to perform the test as well our *HCIPDB01* COBOL program..

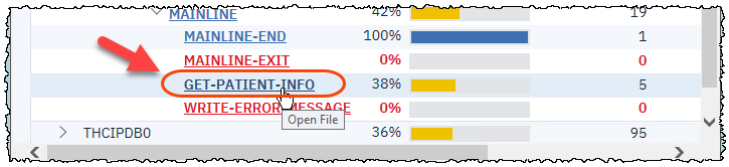
⏭ Since we are just interested in the code coverage of program being tested **click** on **Modules**:



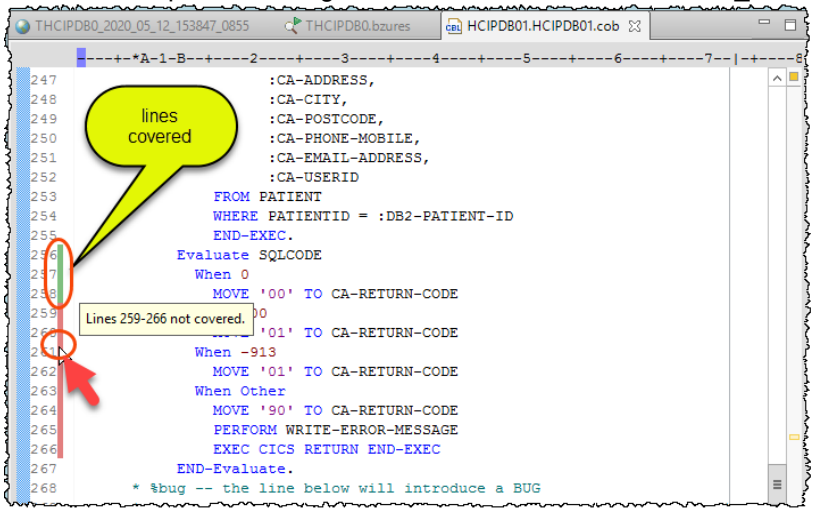
3.5.6 **Expand the nodes** as below to see the COBOL paragraphs.
Notice that *GET-PATIENT-INFO* has 38% of coverage and that *WRITE-ERROR-MESSAGE* was not covered at all on this test case.



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



3.5.8 **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.



3.5.9 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 2.2.20 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 4. 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.

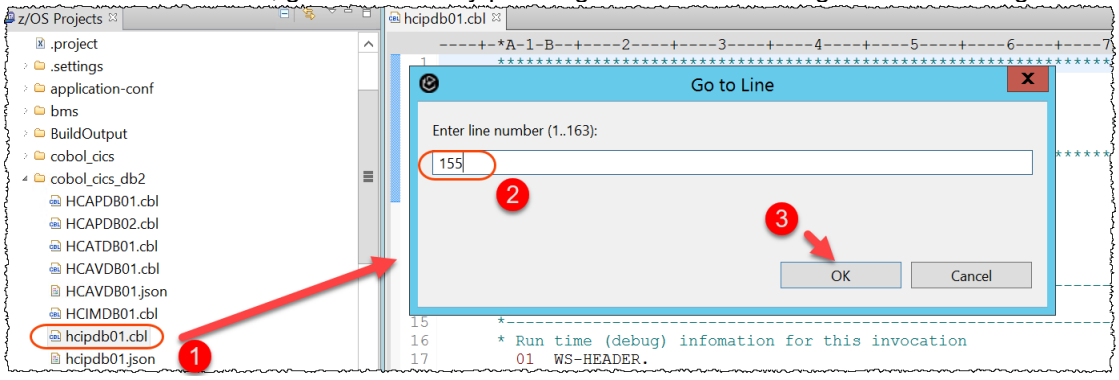
___ 4.1 Modifying the program and introduce a bug

.

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

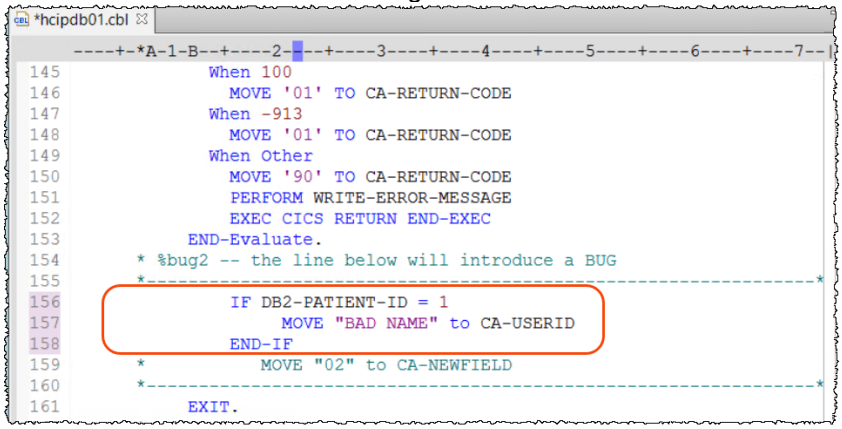
⏭ Open **hcipdb01.cbl** under **cobol_cics_db2** by double clicking on it in the z/OS Projects view.

4.1.2 ⏭ On the editor, go to line 155 by pressing **Ctrl+L** and entering **155** and clicking **OK**.



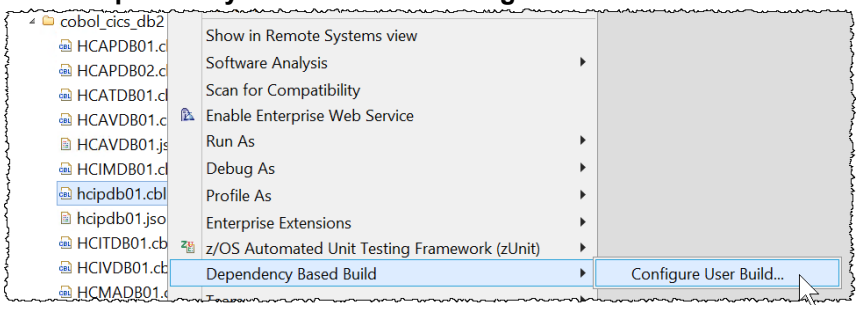
4.1.3 ⏭ Change the lines 156, 157 and 158 removing the * from the statement that moves "**BAD NAME**", Tip -> Could use **Source > Toggle Comment**

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

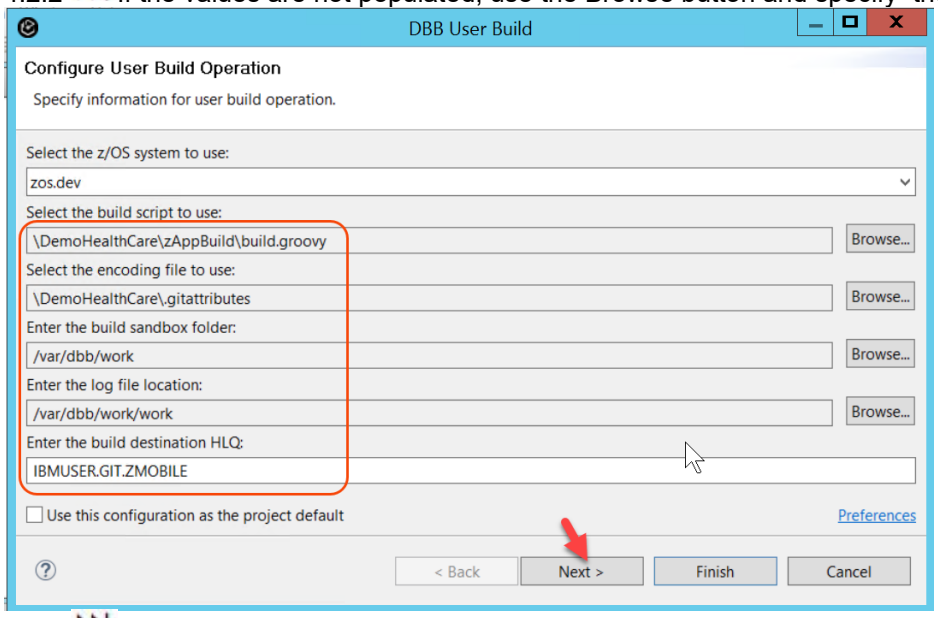


___ 4.2 Rebuilding the changed program without deploying to CICS using DBB

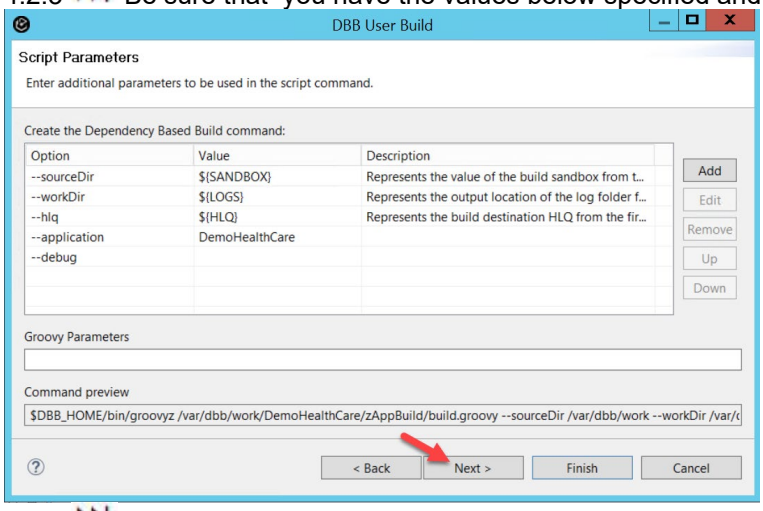
.4.2.1 ⏭ On the z/OS Projects view, right click on **hcipdb01.cbl** and select **Dependency Based Build > Configure User Build**…



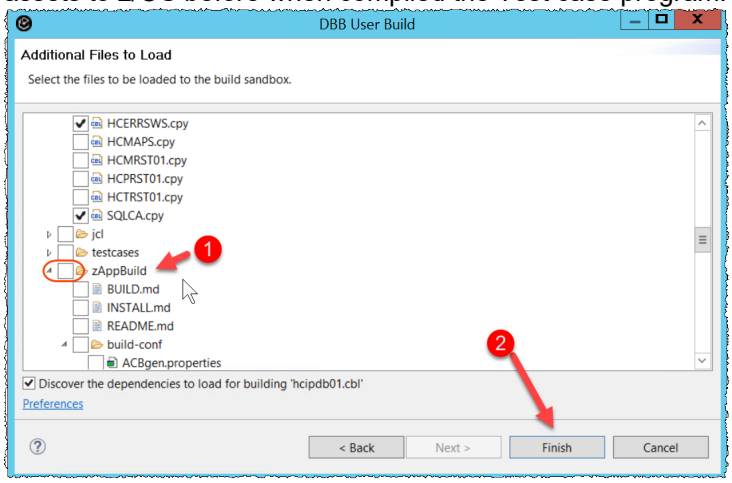
4.2.2 If the values are not populated, use the Browse button and specify the values below and click **Next**



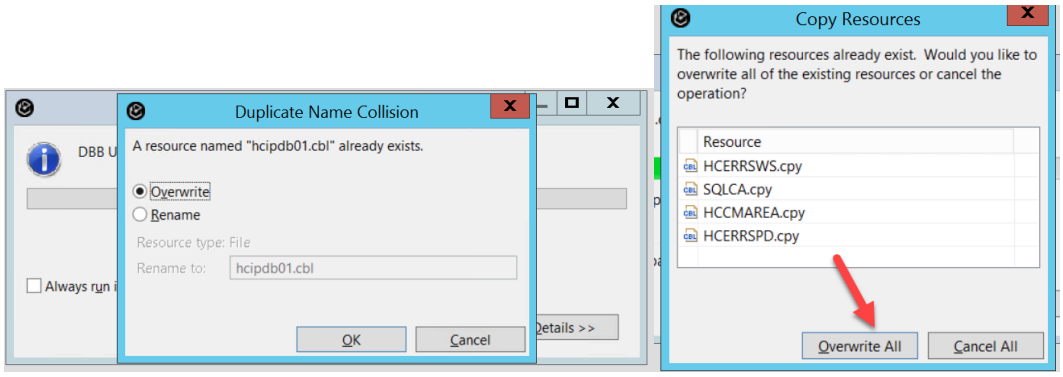
4.2.3 Be sure that you have the values below specified and click **Next**



4.2.4 Be sure that you had Un-select **zAppBuild** and click **Finish** . You have already moved all those assets to z/OS before when compiled the Test case program.

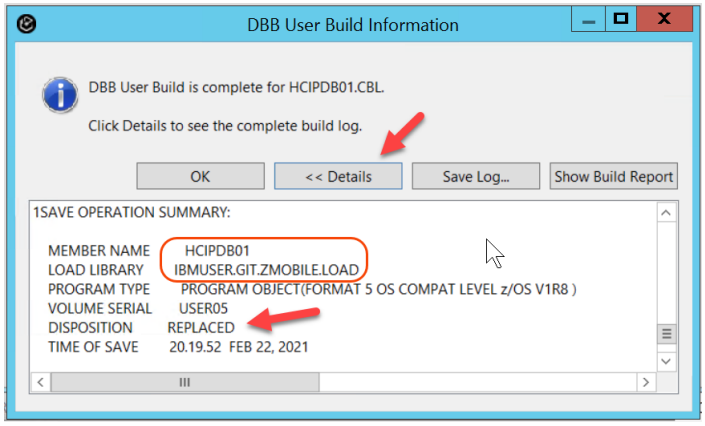


4.2.5 ⏭ Overwrite the code already on z/OS as the example below

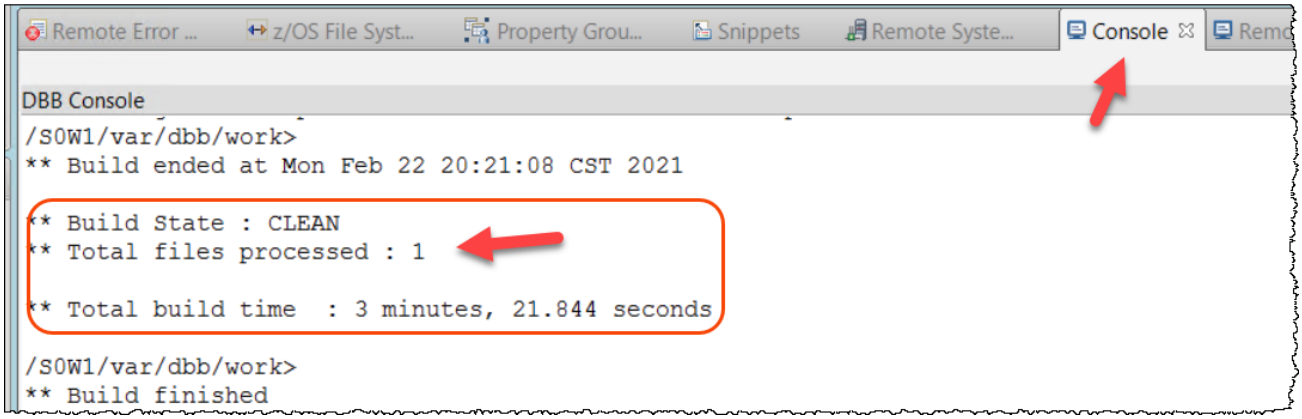


4.2.6 This operation may take up to 4 minutes

⏭ When finished the dialog below will be displayed and you can click on **Details** and **OK** after you verified that the load module was created.



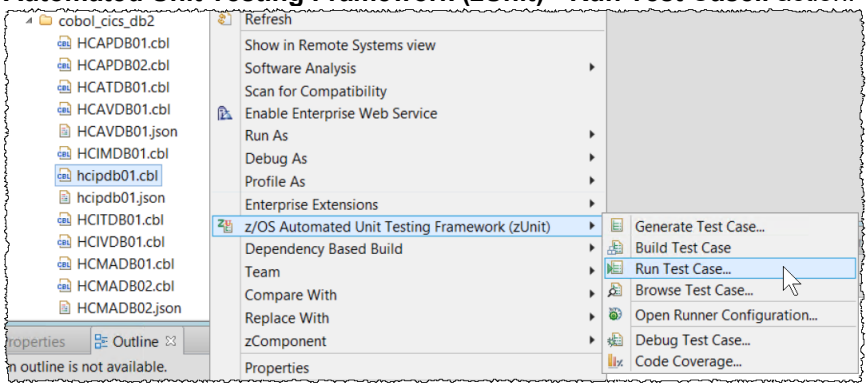
⏭ Click on the ***Console*** view to see the results:



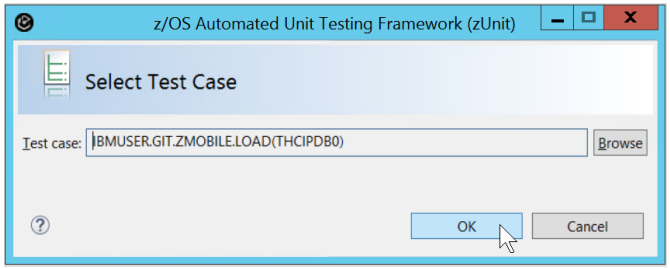
___ 4.3 Running the test case again

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

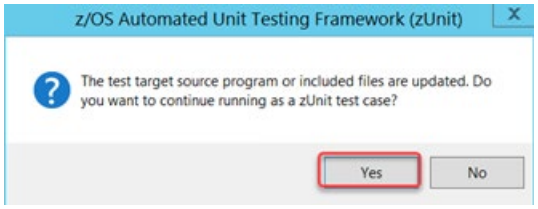
4.3.1 ⏭ On the z/OS Projects view, select **hcipdb01.cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)->Run Test Case..** action.



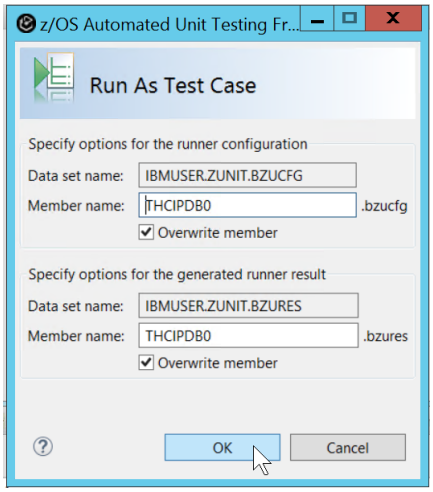
4.3.2 ⏭ Select the test case load module **IBMUSER.GIT.ZMOBILE.LOAD(THCIPDB0)** and click **OK**



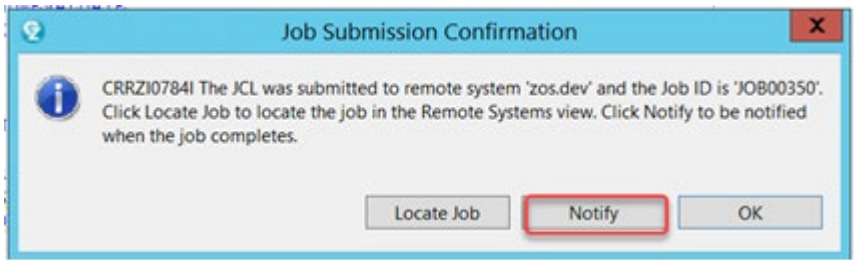
4.3.3 ⏭ Click **Yes** to the following dialog to continue running the test case.



4.3.4 ⏭ Click **OK** to the **Run As Test Case** dialog.

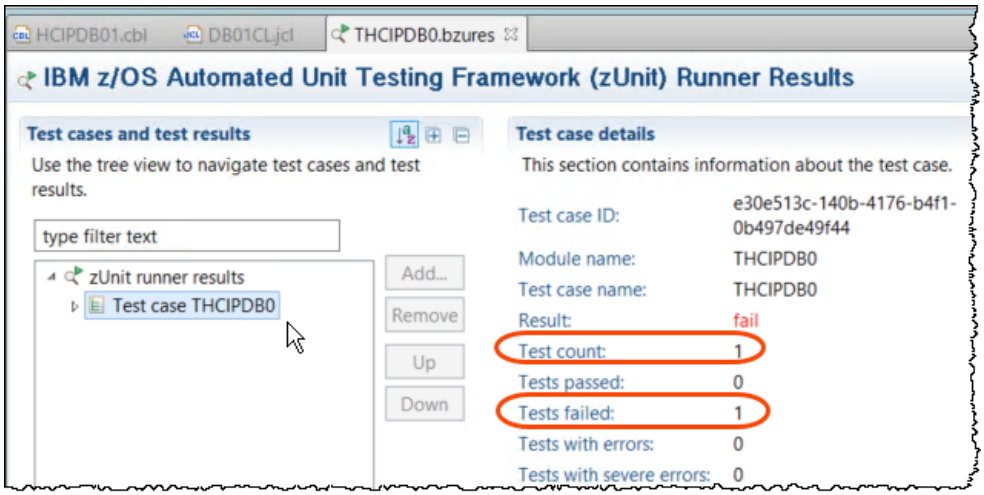


4.3.5 ⏭ Click **Notify** on the Job Submission Confirmation dialog.

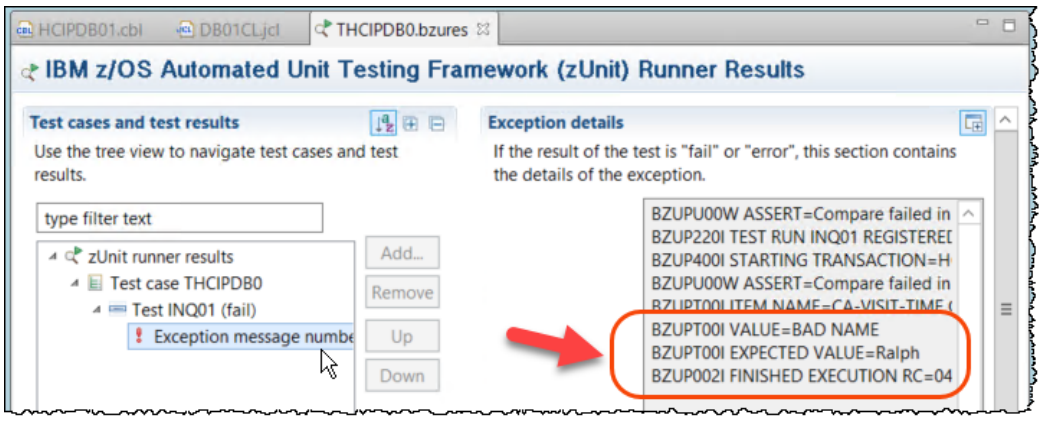


4.3.6 When the test run completes, the test results will be displayed, and it showed 1 test was ran and it failed.

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



4.3.7 ⏭ 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



___ 4.4 Running zUnit test case with debugging

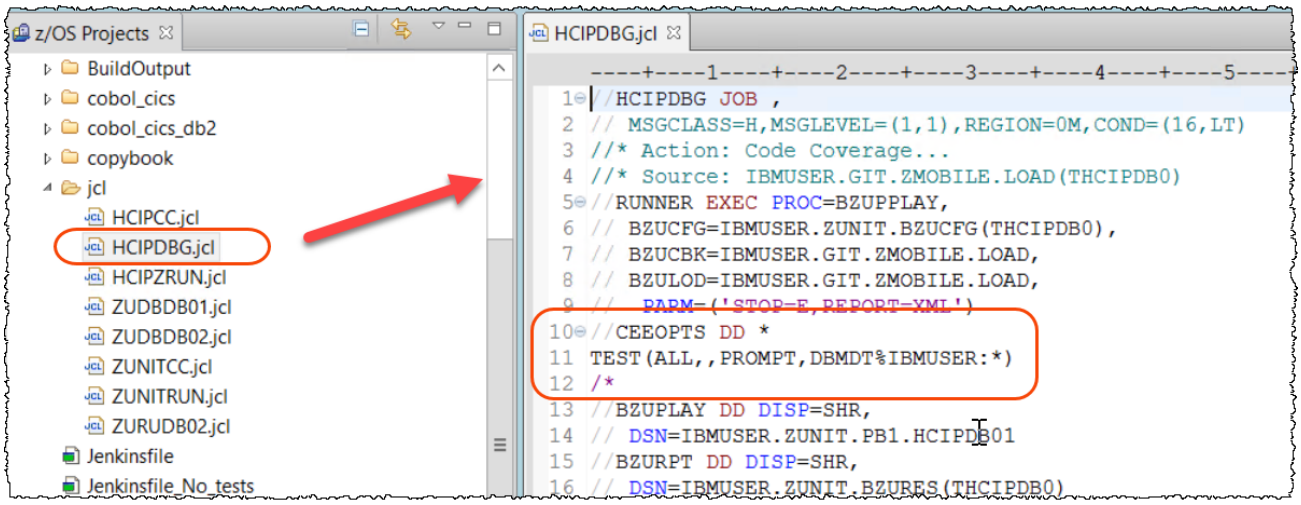
The Debug dialog will debug the test case. Note that will debug the zUnit programs, the generated COBOL and also the COBOL program that you are testing. Below one example.

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

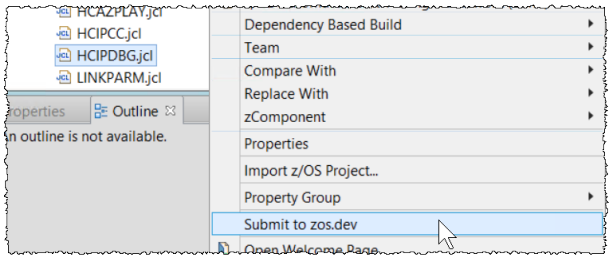
4.4.1 ⏭ Close all active windows by pressing **Ctrl+Shift+F4**

4.4.2 We saved the JCL showed on step 3.4.2 and added the information to run the debug. This JCL can be found at **`jcl/HCIPDBG.jcl`**

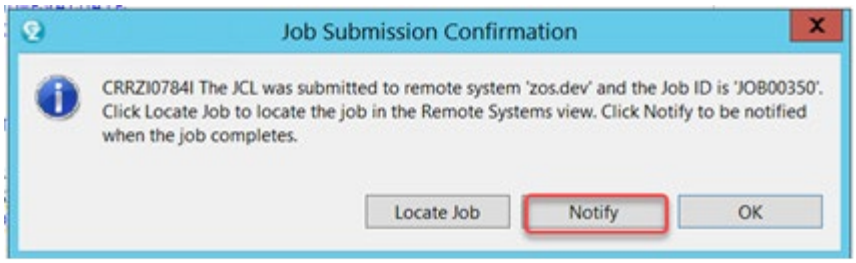
⏭ Double click on **`jcl/HCIPDBG.jcl`** and verify this JCL.



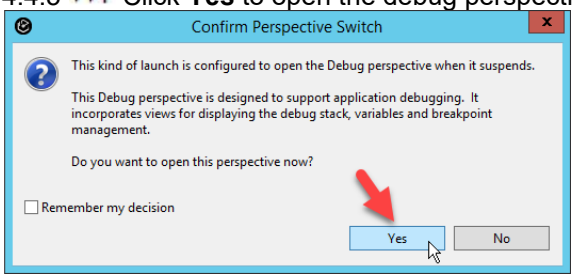
4.4.3 ⏭ Right click on **`HCIPDBG.jcl`** and select Submit to zos.dev



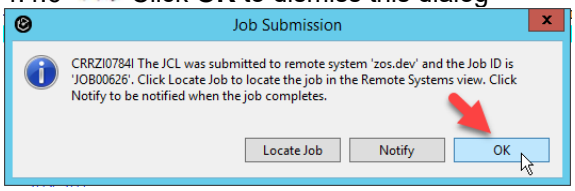
4.4.4 ⏭ Click **Notify** on the Job Submission Confirmation dialog.



4.4.5 Click **Yes** to open the debug perspective.



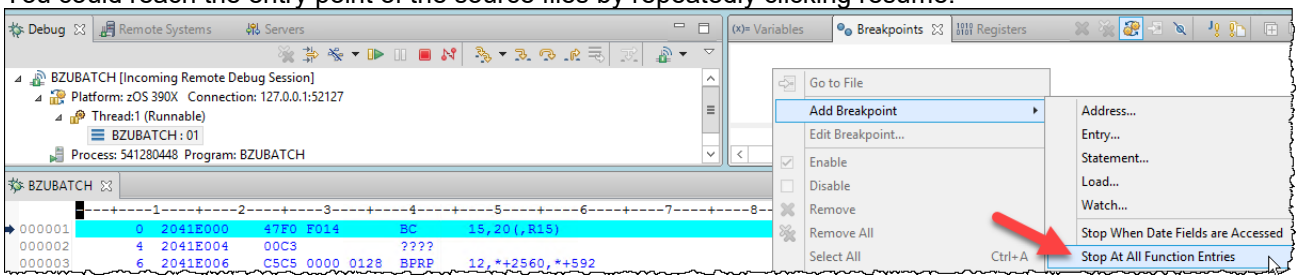
4.4.6 Click **OK** to dismiss this dialog



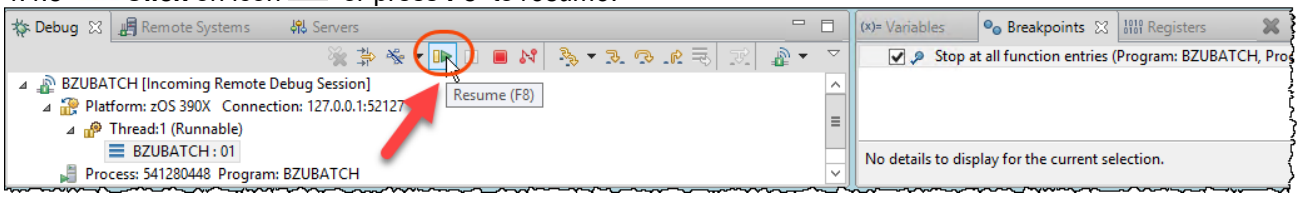
4.4.7 Notice that the first program being debugged is the zUnit engine (BZUBATCH) that is not a COBOL

Using the Breakpoints view, right click on it and be sure that **Add Breakpoints > Stop At All Functions Entries**. is selected, If not you must select it

You could reach the entry point of the source files by repeatedly clicking resume.

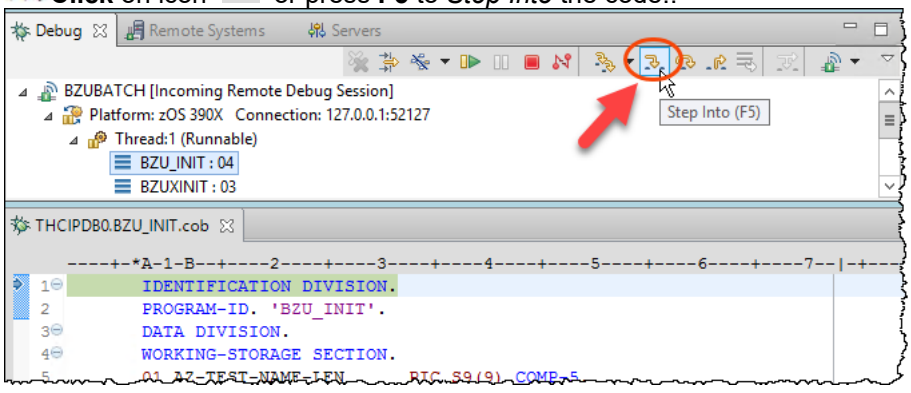


4.4.8 **Click** on icon or press **F8** to *resume*.



4.4.9 This COBOL generated program **BZU_INIT** will start in debug mode..

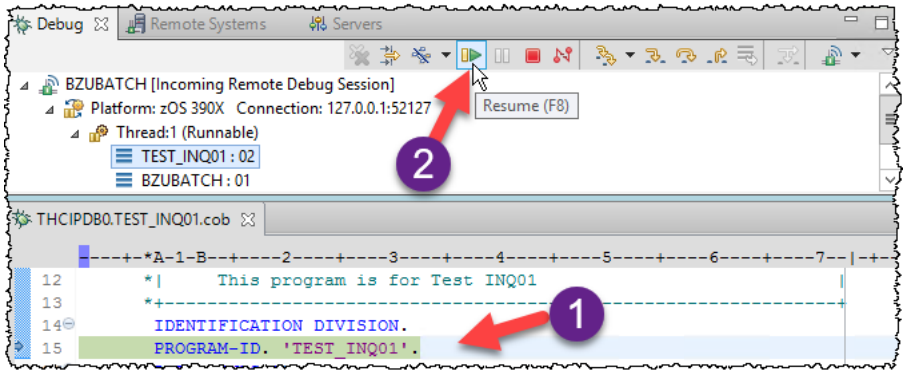
Click on icon or press **F5** to *Step Into* the code.:



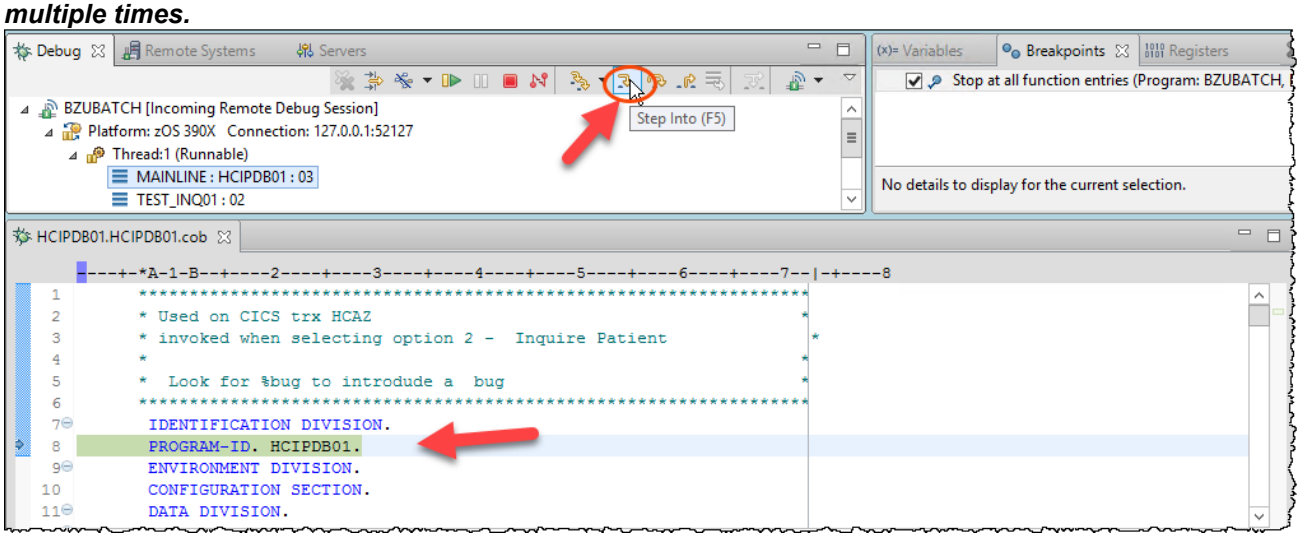
4.4.10 **Click** on icon  or press **F5** to *Step Into* the code until you reach the program **TEST_INQ01**

Click on icon  or press **F8** to *resume*..

Or if you prefer keep clicking on Step Into.

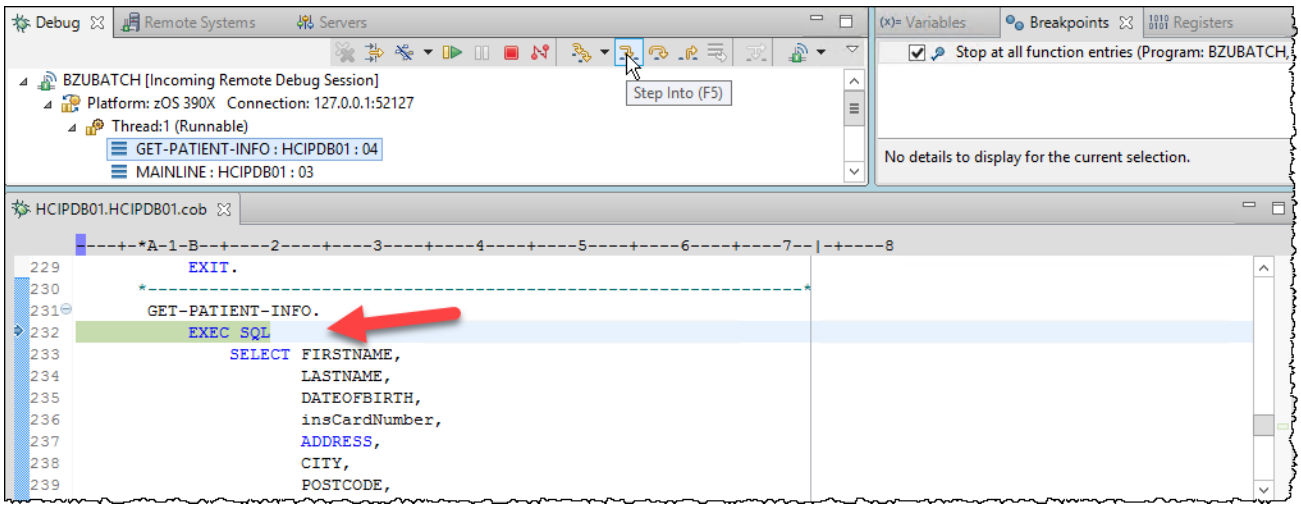


4.4.11 Once the debug stops on your program **HCIPDB01** click on icon  (or **F5)** to *Step Into* the code ***multiple times.***

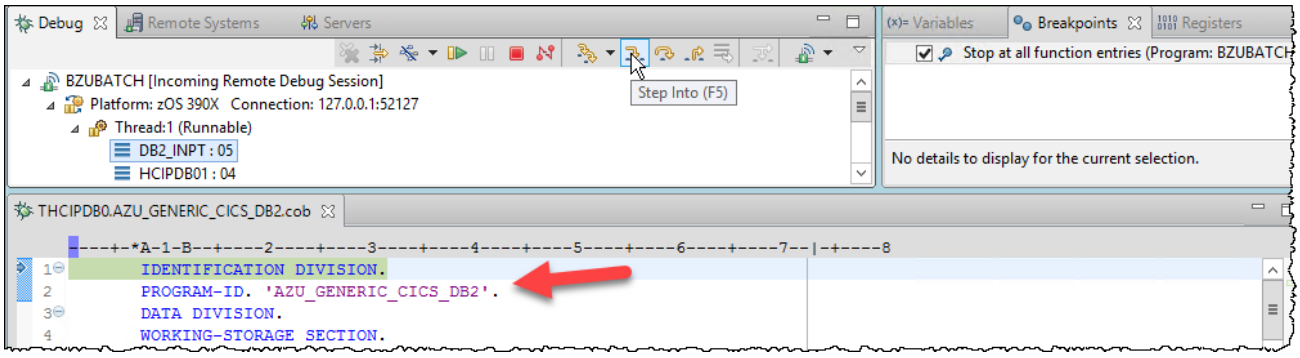


4.4.12 When you see the ***SQL SELECT*** statement you may use the step into to verify that you are using "stubbed code to get the data from DB2..

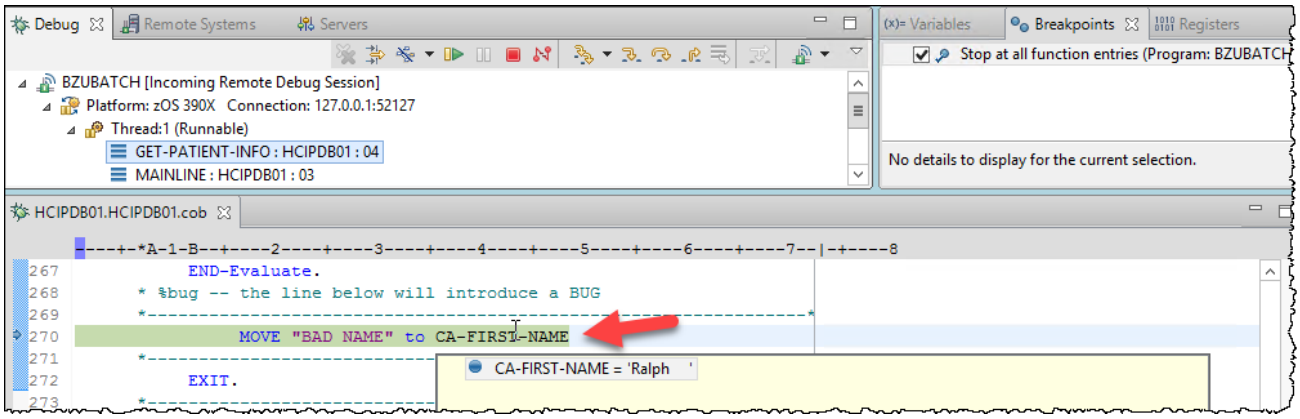
...



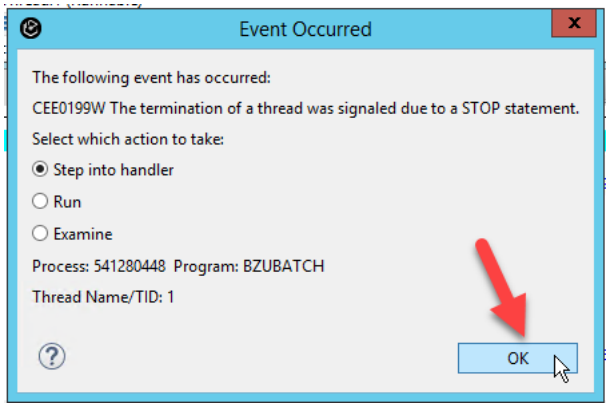
4.4.13 Keep clicking on icon (or **F5)** to *Step Into*
Noticed that the DB2 is not being invoked to access the DB2 table a program named **AZU_GENERIC_CICS_DB2** is invoked to get the test case data that you recorded.



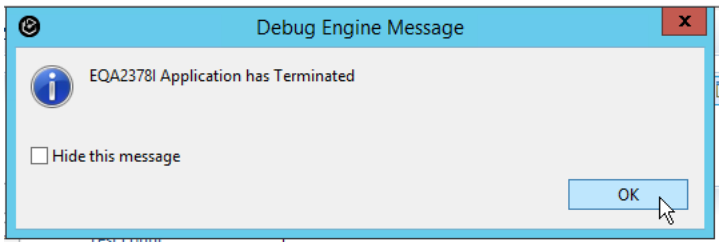
4.4.14 You also can see the BUG that you introduced. Move the mouse to **CA-FIRST-NAME** to see the field value (Ralph).



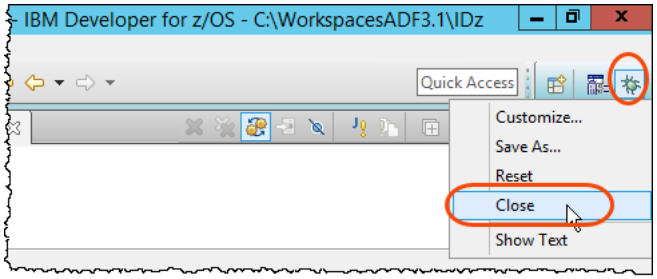
4.4.15 Fell free to continue debugging until the end of the test case execution Or just keep clickin on icon (**F8**) to *resume*..
You will see that the debug ended when you have the dialog below. Click **OK**



4.4.16 Click **OK** to terminate the debug



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



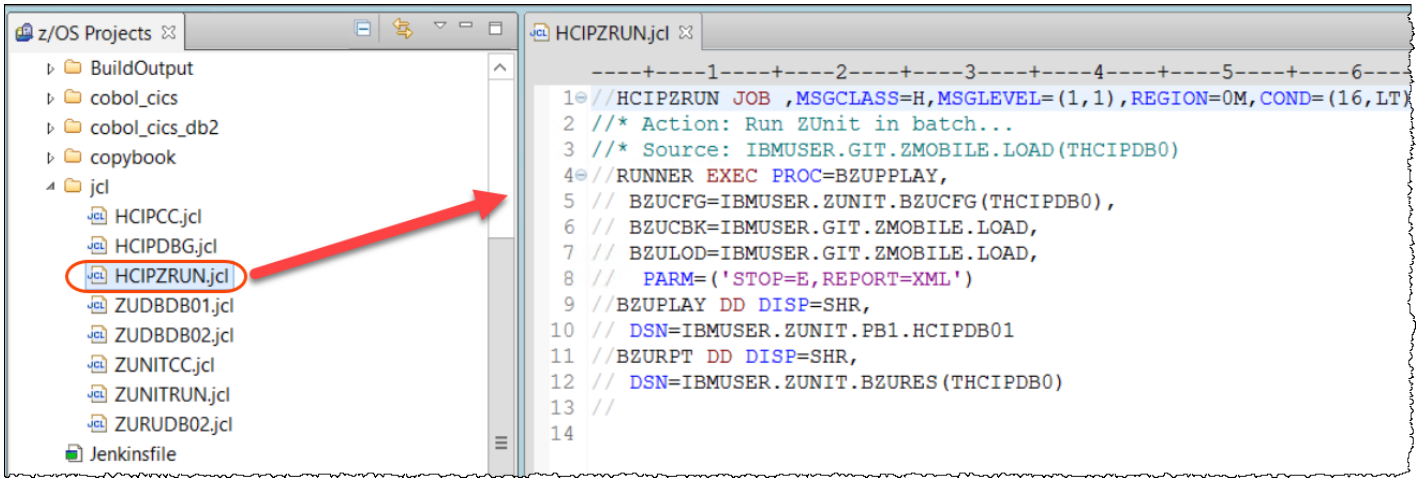
Section 5. Run the unit test from a batch JCL.

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..

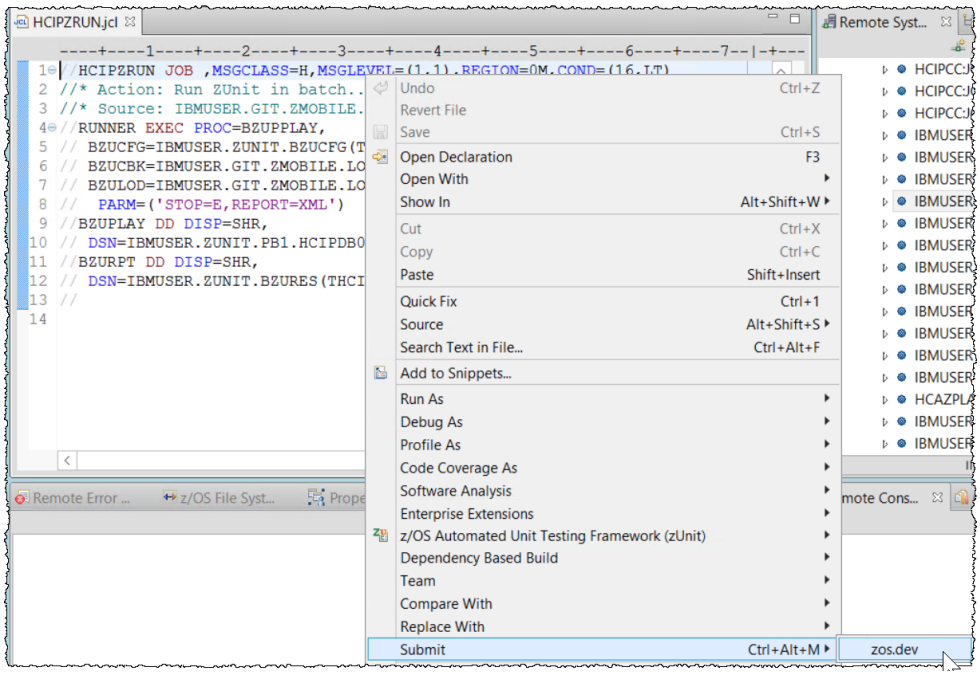
___ 5.1 Running the unit test from a batch JCL

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

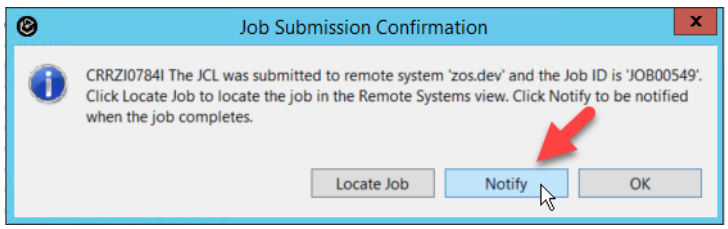
5.1.2 On the IDz *z/OS Projects* view, double click **HCIPZRUN.jcl** to open it.
This JCL will run the test case that you created using a batch job.



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

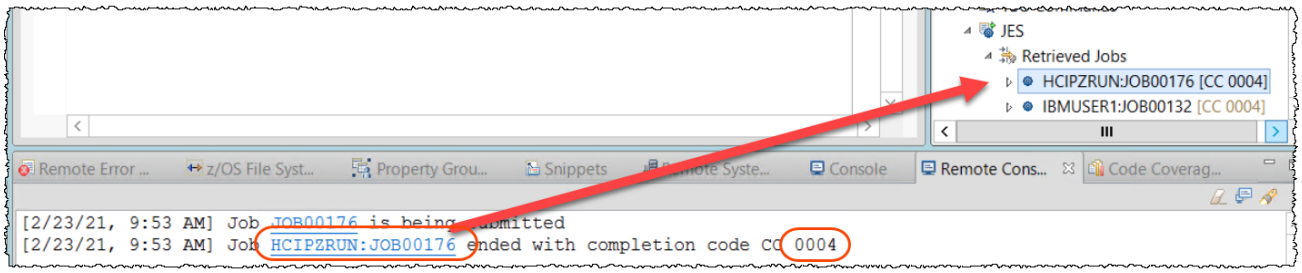


5.1.4 ⏭ Click **Notify** on the Job Submission Confirmation dialog.

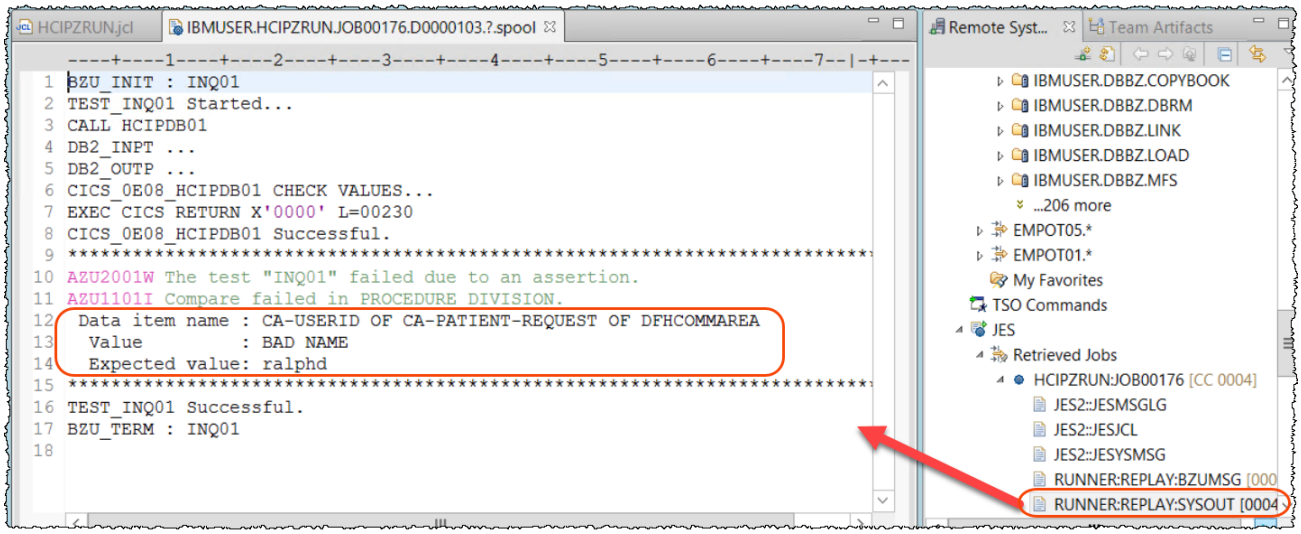


5.1.5 You MUST have **0004** as return code.

⏭ Click on **HCIPZRUN:JOB00xxx**

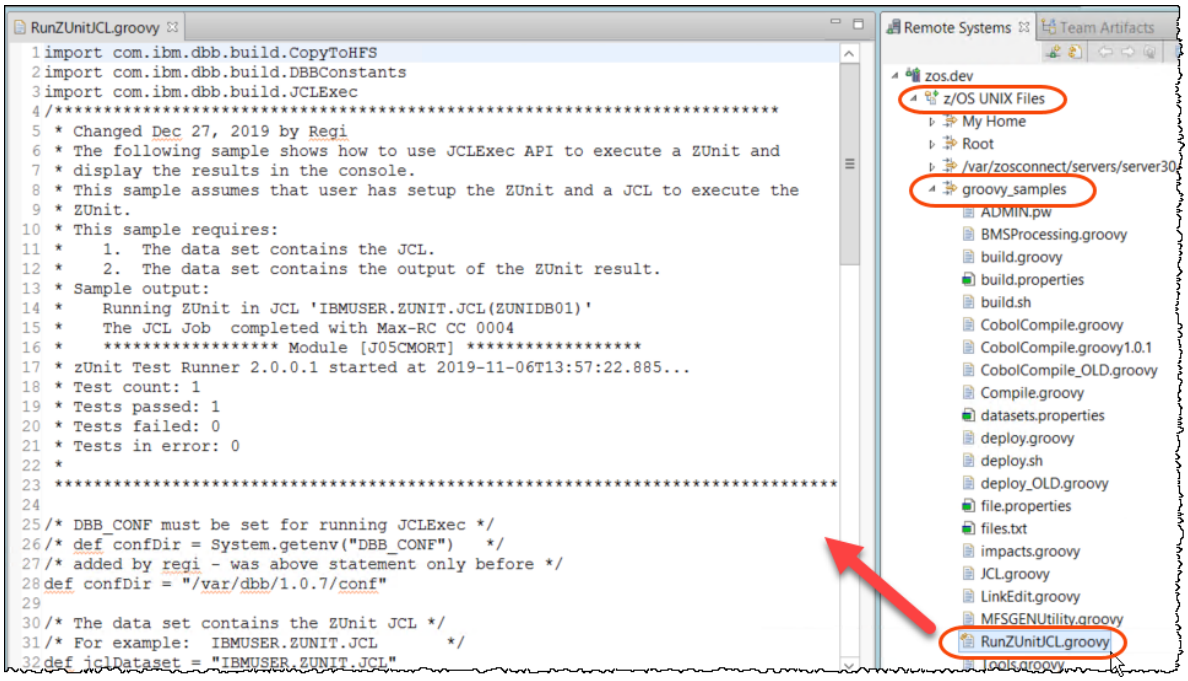


5.1.6 ⏭ Expand **HCIPZRUN:JOB00xxx** and verify the results double clicking on **RUNNER:REPLAY:SYSOUT.**



5.1.7 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.